1-2003

A two-step approach Bayesian network model selection

Michael John Kane

A Two-Step Approach Bayesian Network Model Selection

by

Michael John Kane

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science
in
Electrical Engineering
from
ROCHESTER INSTITUTE OF TECHNOLOGY
August 2003

This thesis is approved:

Dr. Andreas Savakis, Thesis Advisor

Dr. Ferat Sahin, Thesis Committee

Dr. Vincent Amuso, Thesis Committee

Dr. Robert Bowman, Department Head

A Two-Step Approach Bayesian Network Model Selection

by

Michael John Kane

ABSTRACT

It is often desirable to show relationships between unstructured, potentially related data elements, or features, composing a *knowledge database* (KD). By understanding the interaction between these elements, we may gain insight into the underlying process from which the KD is derived, and as a result, we can often model the process. *Bayesian Belief Networks* (BBN) in particular, are adept at modeling knowledge databases for two reasons. The first is that BBNs give a structural representation of data elements through a *directed acyclic graph* (DAG). This ability may make BBNs invaluable in areas such as data mining, where statistical relationships between the features of a traditional database are not apparent. An accurate BBN will clearly show features exerting influences on each other. The second strength of the BBN model is its ability to encode conditional expectations between knowledge database features. This ability facilitates using BBNs as inference engines. Given a set of instantiated elements, BBNs allow us to derive the most statistically likely instantiation of states for elements whose state is unknown. These qualities lend themselves to BBNs being proficient in applications ranging from computer vision to risk-assessment.

In this thesis, two frameworks for BBN structure learning, or model selection, will be compared. The first is the asymptotically correct structure learning algorithm which shows efficient search space *exploration* characteristics. The second takes permutations of global structures in an elitist elimination heuristic search and shows precise search space *exploitation* characteristics. Comparisons between techniques will be presented, paying particular

attention to computational complexity versus model precision. In the elitist elimination technique, comparisons between the Minimum Description Length (MDL) scoring heuristic and the Database probability given Model (DGM) scoring heuristic, will be provided. A comparison between naïve and non-naïve structure learning will be made along with an analysis of the infeasibility of naïve BBN model selection. Finally, an efficient and precise algorithm for learning BBNs, which utilizes both frameworks, will be proposed.

Contents

Chapter 1

Introduction

1.1 Bayes Theorem

Expectation of data given other data arises in many areas of science and engineering. It is often the case that data elements in a data set interact and influence each other in a way that it is difficult to describe with a causal model. This may happen because we are not aware of the relationships between elements, interaction between elements is highly non-linear, or the number of elements would yield an unwieldy model.

In cases where causal models are not preferred, a probabilistic model may be used. Probabilistic models are generally more adept at handling situations where there may be exceptions, doubt and a lack of regularity. In these situations, we can model elements using their independent probabilities or, in cases where there is an interaction between elements,

conditional probabilities.

Bayes Theorem can be interpreted as expressing conditional relationships between elements by using one set of elements as a context or frame of knowledge for another [15]. In this way, the expression $p(A|B)$, the probability of A conditioned on B, specifies the probability of occurrence of A in the context specified by B. As a result, empirical knowledge of elements can be encoded into the conditional probability equation:

$$P(A|B) = \frac{P(A,B)}{P(B)} = \frac{P(A)P(B|A)}{P(B)} \tag{1.1}$$

It can sometimes be assumed that the set of variables that supply the context are independent of each other, that is, for a set of n context random variables in B with $b_i \in B$ Bayes theorem becomes:

$$P(A|B) = \frac{P(A)\prod_{i=1}^{n} P(b_i|A)}{\prod_{i=1}^{n} P(b_i)} \tag{1.2}$$

This is generally referred to as naïve Bayes.

1.2 Bayesian Networks

A Bayesian belief network is defined as the representation of qualitative relationships between variables by a directed acyclic graph (DAG) and the superimposing of a joint probability model on the unknown quantities. This definition implies two major components. The first is the graphical structure, which provides both a visual representation of the nodes

as well as keeping track of those nodes which influence, explain or provide cause for other nodes. The second is the joint probability model used to quantify interrelationships between nodes, which is encoded by Bayes theorem.

1.2.1 Representation

Graphs are often incorporated into statistical modeling. The reasons for this are to show relationships between vertices which cannot be ignored. In BBNs specifically, the graph structure efficiently represents joint probabilities and facilitates inference from observation [15] by showing conditional expectation between elements or *features* of the knowledge base.

Bayesian belief network nodes may be classified in terms of the edges that connected them. A node with an edge running *to* another node indicates a context, condition or assumption for the node which is pointed to. These are called *parent* nodes, and the nodes they point to are their *children* nodes. A node with no parents is referred to as a *root*, *source* or *independent* node while a node with no children may be referred to as a *leaf* or *sink* node.

1.2.2 Statistical Inference

The most common task for BBNs is inference. In these situations, the state of a proper subset of nodes in a network is known and this information is used to find the states of

uninstantiated nodes.

Let a *directed Markov field* over a DAG satisfy the condition that for disjoint random variable sets A, S and B, whenever A and B are separated by S, A and B are *conditionally independent*. Given a DAG with node set V, a set of random variables $X = (X_v : v \in V)$ and joint distributions for X which are *directed Markov* with respect to the graph, the probability of a given node is:

$$p(x) = p(x_v|x_{pa(v)}) \tag{1.3}$$

where x_v is the vth node in the graph and $v_{pa(v)}$ is the set of parents of x_v. The *joint probability distribution* of the model M is

$$p(M) = \prod_{v \in V} p(x_v|x_{pa(v)}) \tag{1.4}$$

If the nodes are *discrete*, i.e. if a node can take on one of a finite number of *states*, we can let $s \in S$ denote configurations of the states for all variables in X. Let the family of a node α, denoted $fa(\alpha)$, be $fa(\alpha) = \{\alpha\} \cup pa(\alpha)$ where $pa(\alpha)$ is the set of parents of α. We can then define the probability of a node as:

$$p(x) = \prod_{s_{fa(v)} \in S_fa(v)} p(x_v(s)|x_{pa(v)}(s)) \tag{1.5}$$

The joint probability distribution of the discrete network model M then becomes:

$$p(M) = \prod_{v \in V} \prod_{x_{fa(r)} \in X_{fa(v)}} p(x_v|x_{pa(v)}) \tag{1.6}$$

It is important to note that in the case where x_n has no parents, the independent probability of the variable is used.

As stated above, in inference, the node instantiation of known node states is used to find unknown node states. Equation 1.3 could imply a system of message passing where "each variable is assigned a simple processor and permitted to pass messages asynchronously with its neighbors until equilibrium is achieved," [15]. With this method, a topological sort could be performed and starting with root nodes, Equation 1.3 could be used to infer node instantiations by selecting the most statistically probable child node state. This process is repeated until the leaf nodes are reached or all uninstantiated nodes are instantiated. In the same way, starting with the leaf nodes we may infer parent nodes, and again, the process is repeated toward the root nodes. This approach is simple and computationally inexpensive, but if the nodes whose states are unknown have *paths*, or undirected sequences of adjacent edges in the graph, between them that are not *blocked*, or all paths cannot be separated by instantiated nodes, it is not guaranteed that will yield the most probable node instantiations.

A more accurate and general approach is implied by Equations 1.4 and 1.6. Rather than finding the most probable local network structure, the entire network structure is examined. In this case, the combination of unknown node states is found and the most likely state configuration is chosen. This provides the most statistically accurate form of inference when compared to other methods.

1.2.3 d-Separation

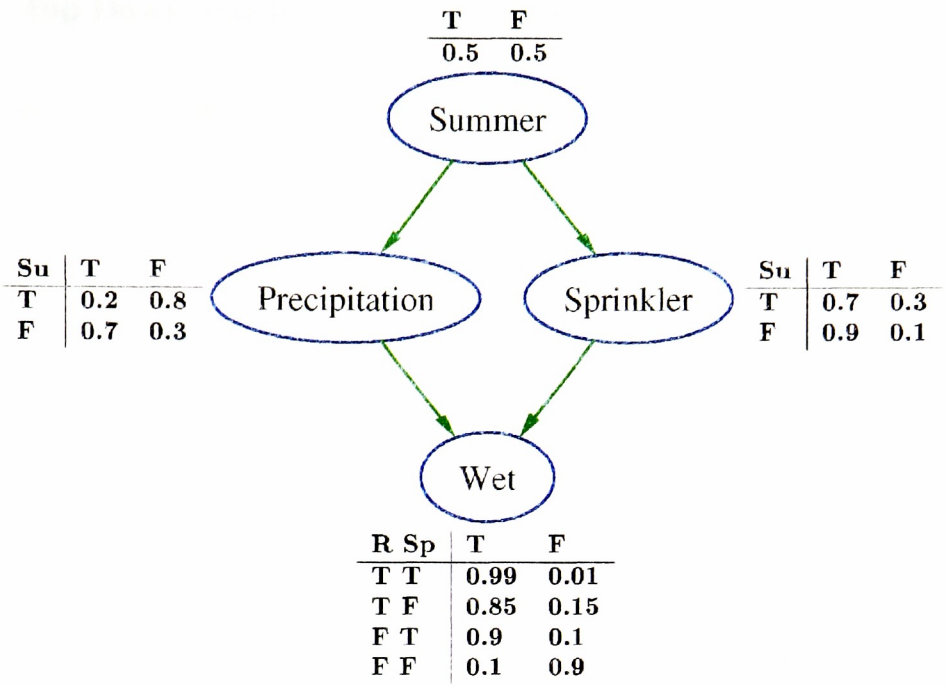
In the previous section, it was indicated that when certain nodes are instantiated, the flow of information between other nodes in the network may be cut off or they may be made statistically independent. If we consider three disjoint node sets, X, Y and Z which are nodes in a DAG, the following conditions determine whether X is independent of Y given Z, or if Z *d-separates* X and Y:

1. A path p with nodes i, m and j contains a chain $i \rightarrow m \rightarrow j$ or a fork $i \leftarrow m \rightarrow j$ such that the middle node m is in Z.

2. p contains an inverted fork, or collider, $i \rightarrow m \leftarrow j$ such that the m is not in Z and no descendants of m are in Z

It is important to note that both of these conditions must be met to fulfill the d-separation criterion.

The meaning of d-separation can be understood intuitively if we attribute a causal meaning to arrows in a BBN. In both causal chains $i \rightarrow m \rightarrow j$ and causal forks $i \leftarrow m \rightarrow j$, i and j are marginally dependent until the state of m is instantiated. After m's state is instantiated the flow of information between i and j is "blocked." On the other hand, with inverted forks $i \rightarrow m \leftarrow j$, the opposite is true. Vertices i and j will become dependent, or connected through an unblocked path once the condition of the middle variable m is known.



Summer	
T	F
0.5	0.5

Su	T	F
T	0.2	0.8
F	0.7	0.3

Su	T	F
T	0.7	0.3
F	0.9	0.1

R	Sp	T	F
T	T	0.99	0.01
T	F	0.85	0.15
F	T	0.9	0.1
F	F	0.1	0.9

Figure 1.1: A simple BBN.

In the context of Figure 1.1 we see that $X = \{Precipitation\}$ and $Y = \{Sprinkler\}$ is d-separated by $Z_1 = \{Summer\}$. This means that knowing whether or not it is Summer will not help us decide whether it is precipitating or the sprinkler is on. Given this case, if we find out if the sprinkler is on, it has no bearing on whether or not it it precipitating given X, Y and Z. If on the other hand we take X, Y and $Z_2 = \{Wet\}$ we see that X and Y are not d-separated. Knowing the instantiation of X will give us information, through explaining away, as to the instantiation of Y, and vice versa, given these sets.

1.2.4 Top Down and Bottom Up Reasoning

An important facet of BBN inference is that a directionality for inference is not implied. This means that we are free to infer parent nodes from children nodes and vice versa. In the former case, by inferring a parent, we are finding a likely "cause" or "explanation" for the parent. In the latter, by inferring a child, an "effect" or "consequence" is found.

Finding conditional expectation can be used to show causality in a probabilistic context. The question arises of whether causality can be shown, as opposed to simply showing correlation, and it has been shown that in some cases causality can be shown. However, the relationships between at least three variables must be measured. One of the variables acts as a "virtual control" for the relationship between the other two.

It is observed in Figure 1.1 that parent nodes may "compete" to justify observed data. In the example, if the grass is wet and it is raining, it is less likely that the sprinkler is on. In these cases, parents are mutually exclusive and often provide an "explanation" for the child.

1.3 Structure Learning

The act of finding the graph structure of a BBN based on an unstructured knowledge database is called *structure learning* or *model selection*. Using the statistical relationships between features in the database, we would like to arrange them in a manner that is ex-

pressive in terms showing conditional relationships. The representation should efficiently show causal links, and elements whose conditional relationship are weak or irrelevant with respect to another need not be shown.

The process of model selection can be automated. Otherwise, BBNs need to be arranged by expert opinion. By automating this process, the graphical structure is quantitatively justified by the fitness function or process from which it is derived. At the same time, by automating this process, we are able to find the structure without the need of a human expert for databases that may have a large number of elements, with relationships that would be difficult for a human expert to derive from simple observation. Structure learning generally falls into one of two categories.

The first category of structure learning is asymptotically correct in that, when the probability distribution of data satisfies certain assumptions, the candidate network will converge to the optimal structure as the number of events it is trained on increases. These algorithms attempt to analyze dependency relationships between nodes to find the best BBN structure. The problem with asymptotically correct BBN model selection is that for large condition sets, conditional independence tests may be unreliable, unless the number of events in the knowledge database is extremely large.

The second category of structure learning uses some scoring method which induces a search space on the space of all BBN structures and then attempts to find an optimum. In the case of BBN model selection, the scoring method is chosen such that an optimum

corresponds to a structure that accurately models the data. This learning category generally faces two obstacles. The first is that the scoring method must be chosen such that the more accurately the structure models the data, the better the candidate structure will score given the scoring method. If this criterion is not met, it is not guaranteed that finding an optimal scoring candidate structure implies finding the best structure. The second obstacle in constructing BBNs from a database of events is there must be enough events in the database to describe the relationships between its features. If this criterion is not met, then there will not be enough information in the database to accurately describe an optimal BBN. In this case the candidate network generally over-fits the data in the knowledge database and describes the database feature relationships and not the underlying features that the database features denote.

1.4 A Note on Naïve Bayes BBNs

To simplify calculations and save space in implementation, child node expectations are sometimes calculated using the naïve Bayes equation. However, if the naïve Bayes assumption of independent parents does not hold, it is clear that the accuracy of the calculation decreases. One then may wonder whether a naïve Bayes model is valid in this case. It turns out for the purpose of inference a naïve Bayes model is competitive with a non-naïve model, and its performance is maximized at the extremes where context variables or

"features" are functionally independent, meaning that the mutual information between the two nodes is at a minimum, or dependent, where there is a high degree of mutual information between the nodes. The worst performance is found between these two extremes [16]. Whether or not the naïve Bayes assumption also holds for structure learning will be addressed in Chapter 6.

1.5 Database Used to Perform Experiments

The database used in this thesis is derived from the ALARM network [8]. This is a BBN constructed from expert opinion as a medical diagnostic alarm message system for patient monitoring. The network is composed of 37 discrete variables which have between two and four states and are connected by 46 directed edges. It is important to note that this network is not a tree or a mixture of trees, but the topology of the graph is sparse.

The database size used in this thesis is $10,000$ events. Two separate databases were used. The first is included with PowerConstructor [3] and the second was generated by Kevin Murphy's Bayes Network Toolbox (BNT) [11]. The total number of directed acyclic graphs given n nodes is given by the following recursive equation:

$$f(n) = \sum_{i=1}^{n} (-1)^{(i+1)} \frac{n!}{(n-i)!i!} 2^{i(n-i)} f(n-i) \tag{1.7}$$

In the case of the ALARM network with 37 nodes the possible number of DAG configuration is very large.

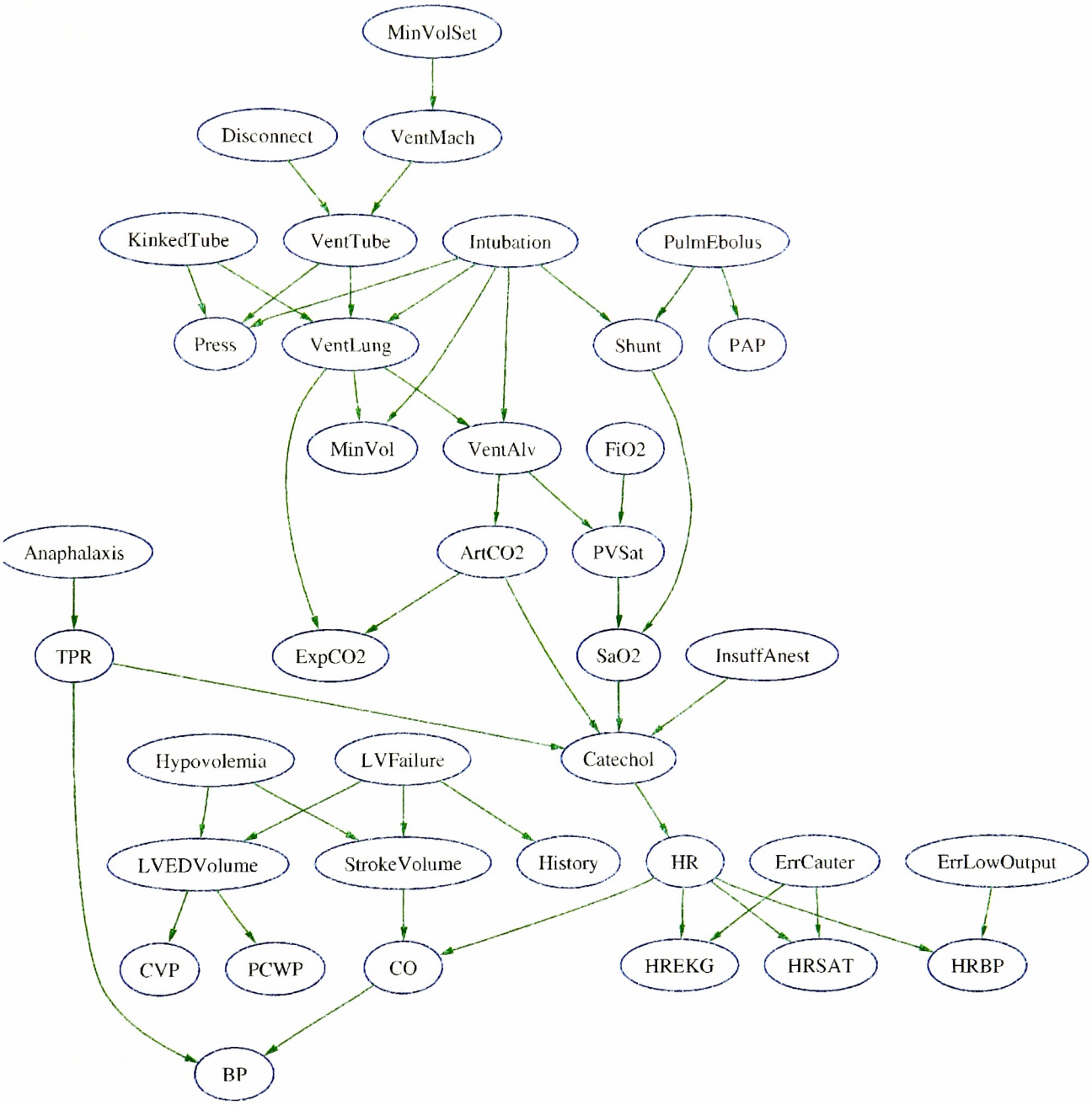


Figure 1.2: The ALARM Network

The total number of unique states for a network consisting of n nodes and $n(s_i)$ states for the ith can be given as:

$$\prod_{i=1}^{n} n(s_i) \tag{1.8}$$

and in the case of the ALARM network approximately $1.7333 \cdot 10^{16}$ unique node instantiations may occur.

1.6 Outline

The remainder of this thesis is concerned with BBN structure learning or model selection in both the non-naïve and naïve case. By exploring the characteristics of structure leaning algorithms, we are able to take advantage of the strengths exhibited by existing algorithms and create a new algorithm that is more more accurate given the computational efficiency involved than any single existing algorithm. Also, by exploring both the non-naïve and naïve case, we are able to assign an appropriate context for either case. In Chapter 2 an overview of asymptotically correct structure learning is provided along with a description of one such algorithm that relies on information theory. The results of using this algorithm with the ALARM network database are given along with a comparison between the ALARM network and the network derived using the information theory asymptotically correct structure learning algorithm.

In Chapter 3 an overview of heuristic search structure learning is provided. This is

a high-level overview of how this class of algorithms work along with examples to more clearly describe their behavior. In Chapter 4 an overview of the fitness functions used in heuristic search structure learning is described along with descriptions of two fitness functions, the Minimum Description Length (MDL) and the Database probability Given a Model (DGM), which are two commonly used heuristic scoring techniques used to quantify candidate structure optimality. In Chapter 5 an approach is proposed which leverages the efficiency of the asymptotically correct information theory approach with the precision of the heuristic search approach. In Chapter 6 results of model selection techniques, as well as a non-naïve versus naïve comparisons, are given. The novel contributions are a new algorithm that combines the asymptotically correct structure learning algorithm with heuristic search structure learning as well as a performance comparison between non-naïve and naïve BBNs. In Chapter 7 the model selection techniques are analyzed, a comparison is given between MDL and DGM, and improvements for the framework are proposed. A proposal for future work, based on the material presented is also provided.

Chapter 2

Asymptotically Correct Structure Learning using Information Theory

An asymptotically correct structure learning algorithm assumes that the probability distributions of features in the knowledge database satisfy certain conditions, such as a high degree of mutual information, indicating a neighbor relationship between nodes. These algorithms attempt to analyze dependency relationships between nodes to find an optimal structure. When these assumptions hold, the structure found converges to the best, or target, structure as the number of events in the knowledge database becomes large.

2.1 The Information Theory Algorithm

The asymptotically correct structure learning algorithm used in this thesis uses the mutual information between nodes for model selection and was proposed and implemented by Cheng et al. [2] in the BN PowerConstructor [4]. The algorithm uses a four-phase belief network construction algorithm.

2.1.1 Drafting

The first phase of this algorithm, referred to as *drafting*, finds nodes which share a high degree of *mutual information*. Mutual information is found using the following equation:

$$I(X_i, X_j) = \sum_{x_i, x_j} P(x_i, x_j) \log \frac{P(x_i, x_j)}{P(x_i)P(x_j)} \tag{2.1}$$

This equation measures the *cross entropy* between nodes. High mutual information values indicate parent-child relationships, although, it does not tell which is the child and which is the parent. Using the mutual information values, an undirected graph is constructed by creating edges between variables where the mutual information is above a threshold value ε indicating a potential parent-child relationship.

If we have N nodes, with maximum number of states r, we will need $\frac{N(N-1)}{2}$ mutual information computations for a complexity of $O(N^2)$. Mutual information will also demand $O(r^2)$ times the operations, such as logarithm, multiplication and division. Sorting the nodes to find those connections that are greater than a given threshold can be done via

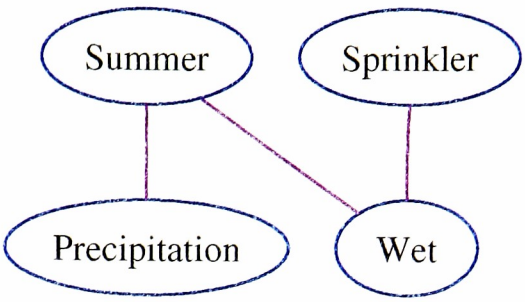


Figure 2.1: The example network after the drafting phase

quicksort which requires $O(NlogN)$ operations for a total complexity of $O(N^2r^2)$ [2].

As an example, suppose we have the wet grass network from Figure 1.1. Also suppose:

$$I(Summer, Precip) \geq I(Summer, Wet) \geq I(Sprinkler, Wet) \tag{2.2}$$

and each of the mutual information values is greater than ε. We can then construct the graph shown in Figure 2.1.

2.1.2 Thickening

The second phase, *thickening*, further connects nodes in the network using the following equation:

$$I(X_i, X_j|C) = \sum_{x_i,x_j,c} P(x_i, x_j, c) \log \frac{P(x_i, x_j|c)}{P(x_i|c)P(x_j|c)} \tag{2.3}$$

Similar to Equation 2.1, the above equation measures the cross entropy between nodes. However, Equation 2.3 takes into account nodes between X_i and X_j when finding mutual

information. This phase connects nodes whose conditional mutual information is above a threshold which may be the threshold from the last phase.

This part of the algorithm must go through all combinations of nodes that are not directly connected finding paths between the two nodes and calculating the mutual information between them. If we let k denote the number of nodes between the two nodes for which mutual information is being found, Equation 2.3 requires $O(r^{k+2})$ basic operations for a potential total of $O(N^2 r^N)$. However, in [4] it is argued that paths between nodes can be optimized with a $O(n)$ algorithm, so the complexity $O(r^N)$ is not included and the complexity is considered $O(N^2)$ [2].

Continuing the example, suppose that

$$I(Wet, Precip|Summer) \geq I(Summer, Sprinkler|Wet) \tag{2.4}$$

and both mutual information values are above ε. The resulting graph is illustrated by Figure 2.2.

2.1.3 Thinning

The third phase of the algorithm is *thinning*. In this phase, edges with node paths between them are temporarily removed and Equation 2.3 is used to test the independence two connected nodes. If the nodes are found to be dependent, the edge is added back, otherwise, the edge is removed permanently. The purpose of this phase is to ensure edges added during

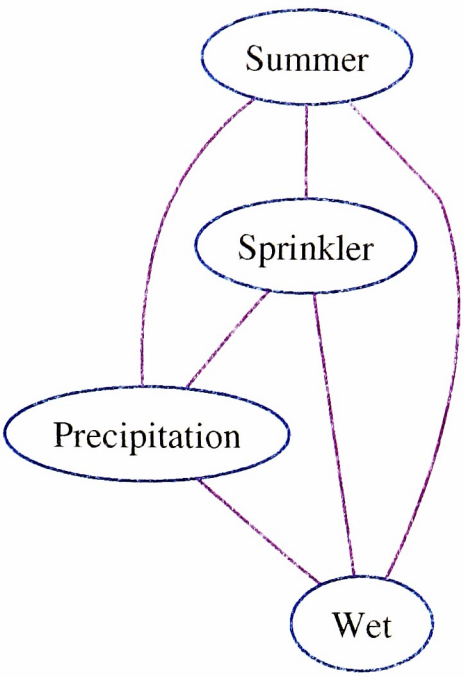


Figure 2.2: The example network after the thickening phase

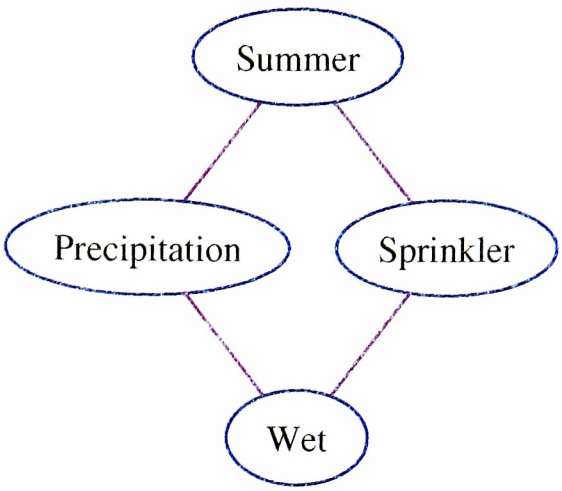


Figure 2.3: The example network after the thinning phase

the last phase did not induce paths between independent nodes. Since this phase tries to remove each edge from the graph, its complexity is $O(N^2)$ [2].

Next in the example, suppose that it is found that the $(Summer, Wet)$ edge along with the $(Precip, Sprinkler)$ edge are not dependent given alternative paths between the nodes. These edges can then be removed and the resulting graph is shown in Figure 2.3.

2.1.4 Edge Orientation

Finally, the edges are given directions. This is accomplished by first finding nodes on the *adjacency* paths between any two nodes $s1$ and $s2$ and putting them into sets $N1$ and $N2$ respectively. Next, neighbors of the nodes in $N1$ that are on the adjacency path between $s1$ and $s2$, and do not belong to $N1$ are put into $N1'$. The same thing is done for neighbors of

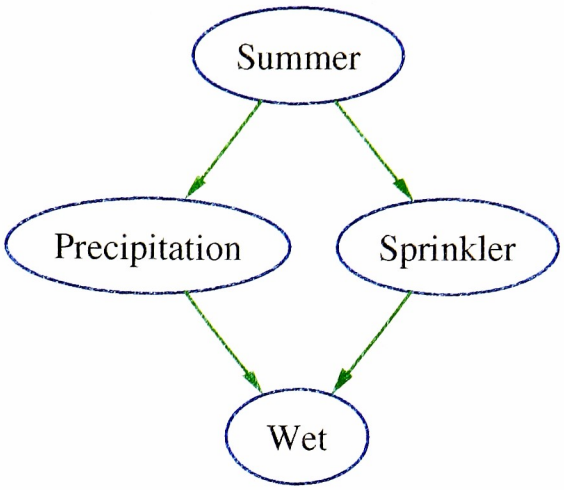


Figure 2.4: The example network after the edge orientation phase

nodes in $N2$. The rest of the algorithm is presented in pseudo-code for readability.

1. if $|N1 + N1'| < |N2 + N2'|$ then set $C = N1 + N1'$ otherwise $C = N2 + N2'$
2. if the conditional independence v between $s1$ and $s2$ is above a threshold ε, go to step 5
3. set $C' = C$ and for each $i \in [1, |C|]$, let $C_i = C$ (the i^{th} node of C), $v_i = I(s1, s2|C_i)$; if $v_i \leq v + \varepsilon$ then $C' = C_i$, let $s1$ and $s2$ be parents of the i^th node of C, else if $v_i < \varepsilon$ go to 5
4. if $|C'| < |C|$ then $C = C'$ and go to step 2
5. start at the beginning until all pairs of nodes are examined
6. for any three nodes a, b and c in C, if a is a parent of b, b and c are adjacent, and a and c are not adjacent and edge (b, c) is not oriented, let b be a parent of c
7. for any edge (a, b) that is not oriented, if there is a directed path from a to b, let a be the parent of b
8. go to step 6 until no more edges can be oriented.

Finally, in the example, the edges are added to the graph using the algorithm described above and the graph in Figure 2.4 is derived.

2.2 Results and Edge Direction Discussion

The results of the information theory approach are shown graphically in Figure 2.5 and a comparison between the information theory approach and the original network is shown graphically in Figure 2.6.

Incorrect edges can be put into one of three categories. The first category consists of edges that point in the wrong direction, we see that this is the case in the edge between the "VentAlv" node and the "Intubation" node in Figure 2.6. This case is the least tolerable since the antecedent parent feature is "explained" by its consequent child feature. In other words, the "causality," "explanation" or justification for a node is backwards.

The second type consists of edges that appear in the candidate structure, but do not appear in the target structure. The edges frequently appear as a grandparent node that points both to a child and a child's child when, in the best case, the grandparent node only points to the child node. In these cases, the indirect statistical influence the grandparent node exerts on the child's child is mistaken for a direct influence and the extra node is added. Although, they are slightly redundant, this is the most tolerable class of incorrect edges. However, if the extra edge precludes the formation of the correct structure by creating a

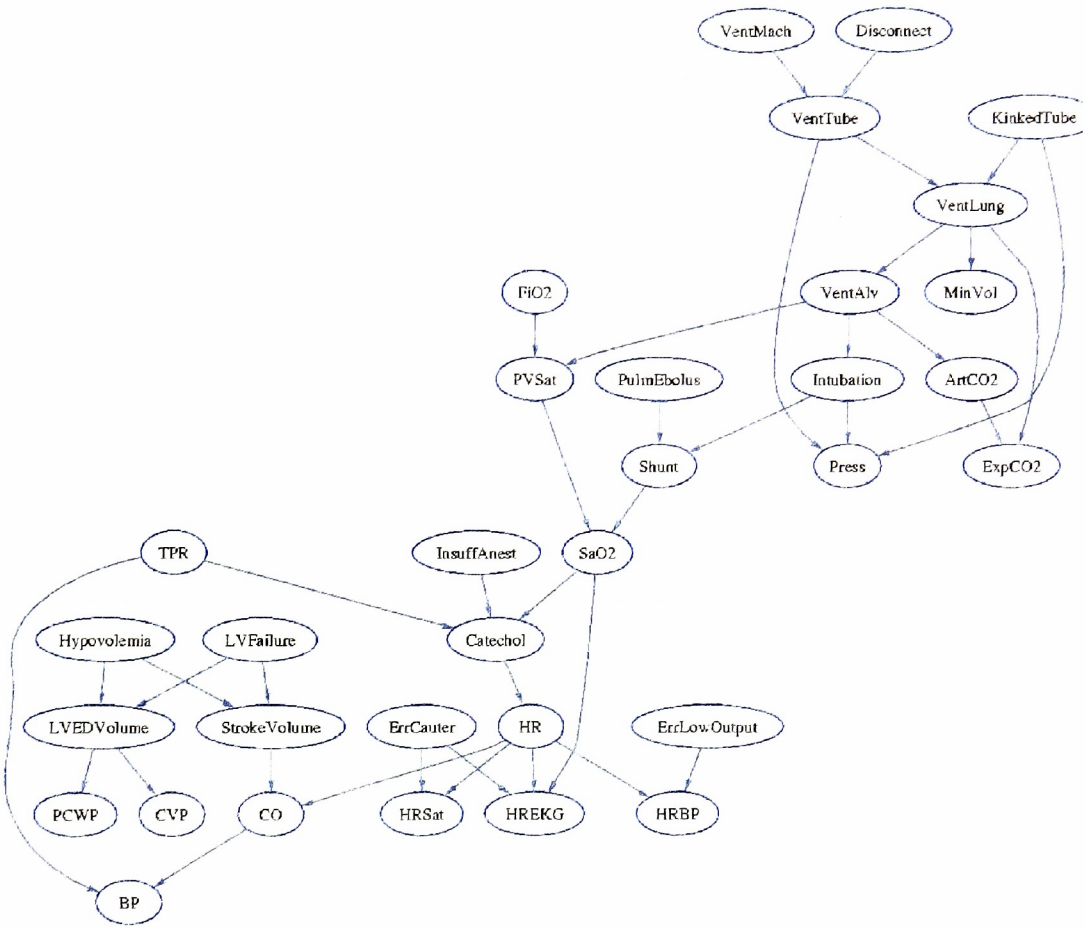


Figure 2.5: The BBN derived from the mutual information algorithm

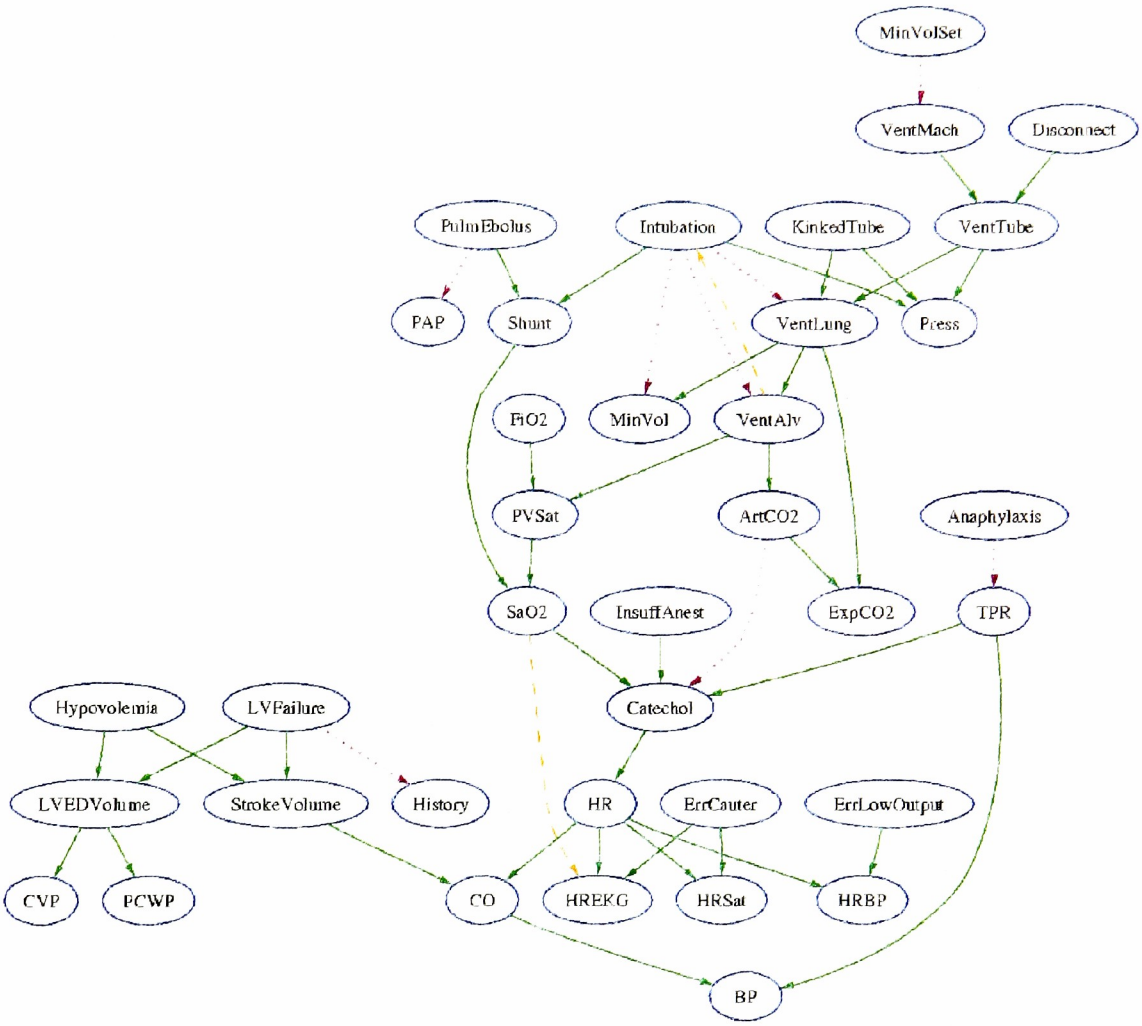


Figure 2.6: The difference between the BBN from the mutual information approach and the ALARM network. Dotted edges indicate edges that are present in the ALARM network, but were not found. Dashed lines indicate extra edges found using the Information Theory algorithm which did not appear in the ALARM network. Solid edges indicate correctly found edges.

cycle should a correct edge be added, these edges become as intolerable as edges in the first category.

The third category consists of edges that appear in the target structure, but do not appear in the candidate structure. Because they fail to show a statistical expectation, these missing edges are not as severe as edges that are backwards, but at the same time they are worse than the extraneous second type.

As can be seen in Figure 2.6 the information theory approach is correct in assigning a majority of edges in the case of the ALARM Network. However, it does make the mistake of assigning one edge in the wrong direction, it misses 8 edges and it assigns two extra edges. With a reasonable computational complexity, it is able to explore a large search space and find a structure that is relatively close to the desired structure.

In the context of efficiently creating a BBN with a reasonable amount of precision, this asymptotically correct structure learning algorithm does well. In fact, in terms of computational complexity, it is currently the most efficient for BBN model selection. However, questions, such as what if our assumptions don't hold or, what if a higher degree of precision is needed, arise. In the next chapter, an alternative model selection frame work is presented which, although not as computationally efficient as this approach, does have the potential to be more precise.

Chapter 3

Heuristic Search Structure Learning

As stated before, heuristic BBN searches use some scoring method which induces a search space on the space of all possible BBNs structures, and then attempts to find a global optimum. This scheme accrues two challenges. The first involves how the search over the search space should be carried out. Ideally, a search framework should guarantee finding the global optimum as efficiently as possible. However, this is not always the case. A search space may also be rife with local optima, and in the case of BBN structure finding, this is generally the case. The second challenge is finding a scoring method where the the optimal score yields the best BBN structure. Finally, we would like the score method to induce a search space that is easily searched.

3.1 The Space Searching Framework

The space searching framework used in this thesis is taken from the "steepest ascent hill climbing" genetic algorithm described by Mitchell [10]. This algorithm encodes a candidate in a binary string and systematically "mutates" or "flips" each bit to create a new candidate variation. After variation is complete, a new set of candidates is created. Each of these candidates is scored and all but the best scoring candidate are discarded. The process is repeated until a desirable candidate is found. This type of algorithm is sometimes referred to as *elitist*, since only the best scoring candidate "survives" and becomes the basis for the next generation of candidates.

To work in the context of BBN structure finding, several aspects of the algorithm were changed. The flow graph, describing this revised process is shown in Figure 3.1. Mitchell [10] describes the first step of the steepest ascent algorithm as, "Choose a candidate solution... at random," and it is often the case that an algorithm is justified as being "genetic" by having a random component to it. In this thesis, however, rather than starting with a random candidate, both a fully disconnected structure as well as the structure derived from the information theory structure finding algorithm were used as starting points. As a consequence, it may be argued that although this is similar to a genetic algorithm, it is in fact not, since it fails the random element criterion.

The next modification made was to the variation portion of the algorithm. Because of the binary representation used to encode a BBN, simply flipping each bit of the binary

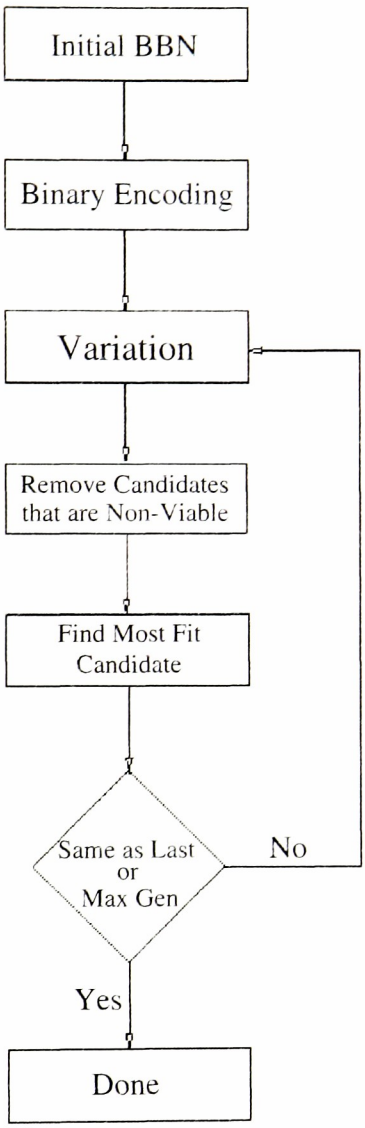


Figure 3.1: The heuristic search flow graph.

representation of a candidate did not create a set of all networks whose only difference was a single edge. Thus, a more sophisticated technique was designed.

Finally, after variation, some of the candidates produced were not DAGs which is a condition required by a BBN. Candidates were produced that were neither members of the search space, nor even the space of all possible BBNs. These candidates are said be non-*viable*, and an extra step has been added after variation that determines candidate viability and removes those candidates from the population of candidates in a given generation that are determined to be non-viable.

The technique described is greedy in that, for each generation, all possible networks that differ by only one edge are taken. Other characteristics of interest in this algorithm included that fact that before being evaluated for fitness, candidates are pre-filtered based on their viability. As this pre-filtering operation is generally less expensive than evaluating the fitness of a candidate, there is some savings, computationally speaking, in this step. Next, the framework supports bottom up, top down and middle out [12] graph searching, and the search is classification is determined by the edges of the initial candidate. If the first basis candidate has no edges, the algorithm becomes bottom up, and adds edges until a local optimum is reached. If the structure is densely connected, with each node having in its set of parents the optimal set of parents, the algorithm becomes top down, removing edges that are not deemed relevant. If a structure beginning from a best guess is used, the technique becomes middle out, and edges are added and removed if they increase the

optimality. In this thesis, the computational complexity of using bottom up verses middle out, where the initial basis is the structure derived from the information theory approach, is explored.

3.2 Binary Representation of a BBN

In selecting a binary representation it is important to encode information about the underlying structure that is relevant. In the case of representing a BBN in a model selection context, what is relevant is the possible edge configurations that are being explored. Thus, relevant information is comprised of the edges between nodes. An encoding scheme should also be compact in that it should minimize irrelevancy and redundancy in its representation, but at the same time, it should not be in such a compact form that decoding needs to be done to manipulate the representation.

In the case of BBNs, the edge between any two given nodes can be described as being in one of three states: no connection; from first to second; and from second to first. The binary representation of a three state system requires two bits. Using this scheme, n nodes will require $n(n-1)$ bits, and since one bit state is wasted per edge description, the scheme's bit efficiency is 75%.

The algorithm for BBN structure encoding is as follows:

```
1. Sort the nodes by their node identifiers.
```

2. Create a binary string s of length $n(n-1)$

3. Create two node iterators it_{n1} and it_{n2} that point to the first and second nodes respectively.

4. Create a binary string iterator it_s that points to the first position of the binary string.

5. If there is a connection between the nodes pointed to by it_{n1} and it_{n2}, then set the position pointed to by it_s to one and then increment it_s. Otherwise, set the position pointed to by it_s along with the position after it to zero and go to step 8.

6. If the node pointed to by it_{n1} is the parent of the node pointed to by it_{n2}, set the position pointed to by it_s to one. Otherwise, set the position pointed to by it_s to zero.

7. Increment it_s.

8. If it_{n2} is at the last position and it_{n1} is at the second to last position, the algorithm is complete.

9. If it_{n2} is at the last position, increment it_{n1}, set it_{n2} to the position after it_{n1} and go to step 5.

10. Otherwise, increment it_{n2} and go to step 5.

The scheme assumes a particular node order as well as a particular order for edge description bits in the binary representation. In this thesis, a lexical sorting of the node identifier is used. The first of the edge description bits describes whether or not the edge is active and the second describes the direction with a one indicating a to-from and a zero indicating a from-to direction. As an example, the binary representation of Figure 1.1 would look like the bit string in Table 3.1

P–SP	P–SU	P–W	SP–SU	SP–W	SU–W
00	10	11	10	11	00

Table 3.1: The binary encoding of figure 1.1

It is important to note that when edge description bits begin with a zero, indicating an inactive edge, the state of the direction bit becomes irrelevant, since, if there is no edge between nodes assigning a direction has no meaning. However, if binary encoding is taken as a *chromosome* in a genetic algorithm, then the direction bit could be interpreted as a *latent* or *dormant gene*.

3.3 Basis Candidate Variation

The purpose of *variation* in the algorithm being presented is to take a current BBN structure and vary a single edge in the network and use varied structure as a new candidate solution. As noted earlier, for any edge between two nodes, there are three possible configurations. Thus, to generate all structures that have exactly one edge difference, there must be two generated structures for each node pair for a total of $N(N-1)$ new candidate structures for N nodes, which are generated during variation for a total of $N(N-1)+1$ members (including the champion from the last generation) of a population for any given generation. The algorithm used in variation is as follows:

```
1. Start with the first edge descriptor (first two bits).
```

2. Copy the bit string twice.

3. If the first bit of the edge descriptor is one, set both bits of the corresponding edge descriptor in the first copy to zero, and set the corresponding edge descriptor of the second copy to the result of the edge descriptor bits exclusive or ''01''.

4. If the first bit of the edge descriptor is zero, set both bits of the corresponding edge descriptor in the first copy to ''11'', and set the corresponding edge descriptor in the second copy to ''10''.

5. If the current position is at the last edge descriptor, the algorithm should terminate, otherwise, move to the next edge descriptor an go to step 3.

Table 3.2 provides an example of the variation algorithm applied to the BBN in figure 1.1.

3.4 Determining Candidate Viability

It may be noted that the variation algorithm does not ensure the binary representation of a DAG. In Table 3.2, the bit string "00 10 11 10 11 11" does not describe a DAG, and, therefore, does not meet the requirements of a BBN. This candidate may be described as non-viable. An efficient method should be chosen to determine candidate viability and those candidates determined to be non-viable should be removed from the population before determining the fitness of members of the population.

The first method that could be used to determine viability is to transform the binary representation of a candidate back into its graph representation. Then, a depth or breadth

00	10	11	10	11	00
10	10	11	10	11	00
01	10	11	10	11	00
00	11	11	10	11	00
00	00	11	10	11	00
00	10	10	10	11	00
00	10	00	10	11	00
00	10	11	11	11	00
00	10	11	00	11	00
00	10	11	10	10	00
00	10	11	10	00	00
00	10	11	10	11	10
00	10	11	10	11	11

Table 3.2: The variation of the binary encoding of figure 1.1

first search could be performed starting at each node, looking for the original node in the list of descendants. For a graph with V vertices and E edges, the computational complexity of a breadth or depth first search beginning at a given node is $O(E + V)$, but since the search needs to be performed for each node, the complexity becomes $O(VE + V^2)$.

The algorithm to create the edges in a graph from its binary string encoding is as follows:

1. Sort the nodes by their node identifiers.
2. Create two node iterators it_{n1} and it_{n2} that point to the first and second nodes respectively.
3. Create a binary string iterator it_s that point to the first position of the binary string.
4. If the values pointed to by it_s is zero and the value pointed to by the position after it_s is zero, go to step 7.
5. If the values pointed to by it_s is one and the value pointed to by the position after it_s is one, make the node pointed to by it_{n1} the parent of the node pointed to by it_{n2} and go to step 7.
6. If the values pointed to by it_s is one and the value pointed to by the position after it_s is zero, make the node pointed to by it_{n1} the child of the node pointed to by it_{n2} and go to step 7.
7. If it_{n2} is at the last position and it_{n1} is at the second to last position, the algorithm is complete.
8. If it_{n2} is at the last position, increment it_{n1}, set it_{n2} to the position after it_{n1}, and go to step 10.
9. Increment it_{n2}.
10. Increment it_s by two.

	P	Sp	Su	W
P	0	0	0	1
Sp	0	0	0	1
Su	1	1	0	0
W	0	0	0	0

Table 3.3: The adjacency matrix for figure 1.1

```
11. Go to step 4.
```

Another method that could be used to determine candidate viability is to attempt to perform a topological sort of the graph. As with the first method, the binary representation of the candidate is transformed back to its graph representation. However, rather than doing depth first searches on each of the nodes, a topological sort is performed. The topological sort creates a linear ordering of a set of vertices in a graph, such that if a directed edge from vertex v_1 to vertex v_2, (v_1, v_2) appears in the graph, then v_1 will come before v_2 in the ordering. If the graph is cyclic, the sort will fail, otherwise the sort will succeed. For a graph with V vertices and E edges, the computational complexity for the topological sort is $O(V + E)$ giving it significant complexity efficiency when compared to the first method.

A third method makes use of the properties of the *adjacency matrix* representation of the candidate. This representation uses rows and columns corresponding to vertices and "0" and "1" values at matrix indices to describe the directed edges between vertices. An

example of the adjacency matrix of Figure 1.1 may look like Table 3.3.

In the case of adjacency matrices, the *spectrum* of the matrix is the set of its eigenvalues and the *spectral radius* of a matrix denotes the largest absolute value of an eigenvalue. It has been noted [6] that if the adjacency matrix does not contain *any* cycles, then it is "permutationally similar" to an upper triangular matrix which will have a spectral radius of zero. Therefore, we can test candidate viability by first transcoding from the binary representation of the graph to the adjacency matrix representation and then find the spectral radius of the adjacency matrix. If the spectral radius is greater than zero the corresponding graph must be cyclic and is not a viable candidate.

The algorithm for a graph with N nodes, with binary encoding vector v and a zero indexed adjacency matrix A is as follows:

```
i_v = 0
for i = 1 : n
   for j = 0 : n − 1
      if v[i_v] is zero
         A[i, j] = 0
         A[j, i] = 0
         i_v = i_v + 2
      else
      i_v = i_v + 1
      if v[i_v] is zero
         A[i, j] = 0
         A[j, i] = 1
      else
         A[i, j] = 1
         A[j, i] = 0
      i_v = i_v + 1
```

As noted before, an adjacency matrix describing a DAG can be transformed into an upper triangular matrix. Thus, rather than finding the eigenvalues for the adjacency matrix, an LU decomposition can be performed on a candidate matrix. If all members of the diagonal are zero, we can say that the eigenvalues will also be zero and the matrix describes a DAG and the candidate structure is viable. The computational complexity incurred by this technique, which is the same as that for the LU decomposition, is $O(n^2)$.

3.5 Finding the Most Fit Candidate

After a population of viable candidates has been determined, the most "fit" candidate is found. This step requires defining a *fitness function* that evaluates the optimality of individual BBNs. Each of the candidates scores are recorded based on the fitness function. Next, the most fit candidate is found and all but this candidate are discarded. If we let the complexity of finding candidates fitness be $O(F)$, then the complexity of this step of the algorithm is $O(FN)$ where N is the number of candidates for a given generation.

At this point in the heuristic search, it is determined whether the survivor of the current generation is the same as that for the previous generation. If it is, then a local optimum has been found and the search terminates, returning the survivor. If it has been determined that the maximum number of generations allowed has been reached, the search will also be terminated. Otherwise, this generation's survivor goes back to the variation step, and the

process continues.

It may be noted that the framework presented provides a general method of solution, not just for model selection, but for a much larger class of problems. Namely, in problem where a candidate solution can be encoded and evaluation by some function. Generally speaking, we see that it is only the fitness function that defines the context for the optimization problem and, the framework provides a context-independent method of space searching. As the framework has been described, we can now move on to the actual fitness function which will describe the optimality of candidates within the framework.

Chapter 4

Fitness Functions

The fitness function is responsible for quantifying candidate optimality. Therefore, in the case of BBN model selection, it is imperative that the fitness function be able to distinguish between candidates that accurately model the knowledge database and those that do not. A secondary responsibility for the fitness function becomes apparent with the realization that it is the fitness function that induces the search space on the space of all BBN structures. Thus, it is desirable to find a fitness function that minimizes the occurrence of local optima, making a global optimum easier to find.

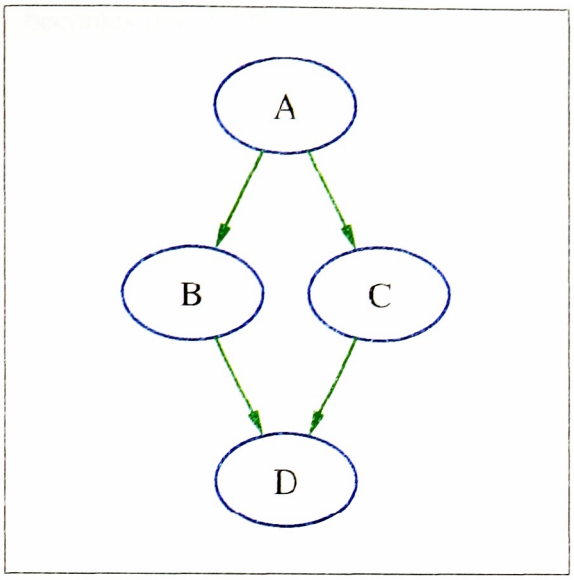
Generally speaking, there are two opposing goals in scoring a BBN with respect to the edges that are added. First, we would like the BBN to be as simple as possible. This means that we would like the network to be sparse. Features that have little or no ability to provide context to children nodes should not be designated as parents. Also, the more sparse the

network, the easier it is to see the significant causal relationships. It has also been found that with BBNs, the degree of connectivity is closely related to the computational complexity, both in space and time, of using the network [9]. Secondly, we want the network to be as accurate as possible with respect to its node-state probabilities. Often, as more context features are added, the precision becomes better, facilitating accurate inference. In most scoring schemes, these two goals are given quantitative terms which somehow counteract each other so that as precision increases, the corresponding precision term becomes "better" and at the same time as the network becomes less sparse, the corresponding complexity term becomes "worse." In this way, the benefit of increasing precision is mitigated with the cost of network complexity.

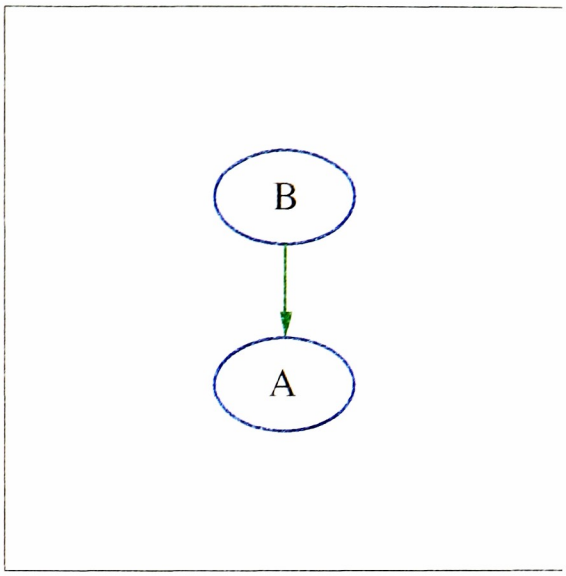
4.1 Joint Network Probability and Confidence in the Naïve Bayes and Non-Naïve Bayes Cases

Taking as an example, the BBN described by Figure 4.1 (a), and using Equation 1.4, we see the joint probability for this distribution is:

$$p(M) = p(A)p(B|A)p(C|A)p(D|B,C) \tag{4.1}$$



(a)



(b)

Figure 4.1:

Taking an even simpler example, as in Figure 4.1 (b) we can explore the joint probability for a graph. Again, using Equation 1.4, we see that the joint probability for this network is:

$$p(M) = p(A)p(B|A) \tag{4.2}$$

If we further constrain the situation by holding the state instantiation of node A to a_i where $a_i \in S_A$ and S_A is the set of possible node instantiations of node A, the equation becomes:

$$p(M_{a_i}) = p(a_i)p(a_i|B) \tag{4.3}$$

If we finally limit the number of possible node instantiations of node A to two states a_1 and a_2, and limit the node B to one possible node instantiation b_1, the model probability

becomes $p(a_1|b_1)p(a_2|b_1)$. Next, taking into account that for n possible states of $b_j \in S_B$:

$$\sum_{k=1}^{n} p(A|b_k) = 1 \tag{4.4}$$

we know that $p(A)$, for the constrained model we have $p(a_1|b_1)$ and $1 - p(a_1|b_1)$ for the possible node state probabilities. The joint probability of this constrained network will be:

$$(p(b_1))^2 p(a_1|b_1)(1 - p(a_1|b_1) \tag{4.5}$$

Disregarding the independent probability $p(b_1)$, so that the term of interest becomes $p(a_1|b_1)(1 - p(a_1|b_1))$, the expression can be visualized by Figure 4.2. One of the node-state probabilities x lies between $0 \leq x \leq 0.5$ while the other node probability lies at $1 - x$ or the reflection of the point across the function $x = 0.5$.

It is important to note in this graph, that the maximum occurs as both node-state probabilities approach 0.5. However, in finding an optimal structure for a BBN, we actually want the individual node-state probabilities to be as far from 0.5 on the graph as possible. In inference, a child node state probability near 0.5 indicates that given a context provided by a parent, only 50% certainty in that particular node instantiation is given. This indicates a high degree of uncertainty in the child instantiation. We would much rather see values near the extremes, approaching 1.0 and 0.0, which indicate that given the knowledge database, the parents do well in providing a context for a node instantiation. From this perspective, we see that functionally, the point of the parent node is to *decorrelate* the node-state probabilities of its children by assigning a context.

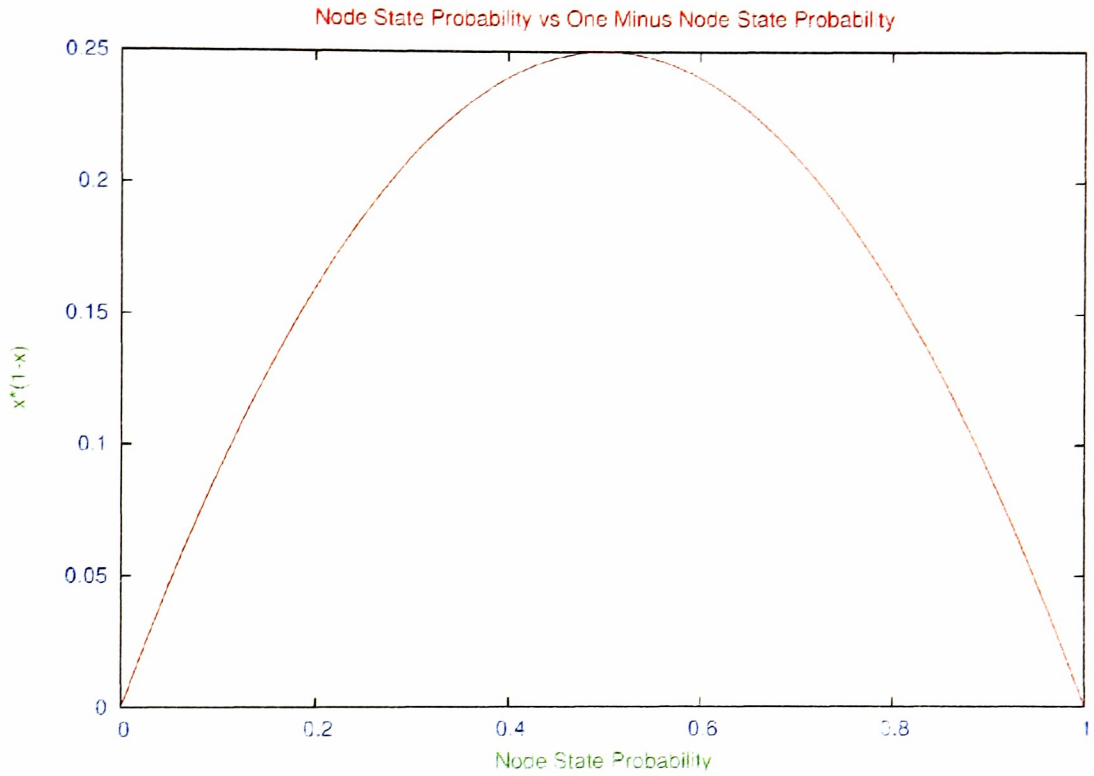


Figure 4.2: The joint node probability distribution function of a two-state child with a single one-state parent.

The problem remains of how to actually quantify our certainty in a network. This thesis uses a variation on the calculation of the joint probability of a network. Let $x \in X$ be the set of nodes in a network with x_i denoting the ith node, and let k denote the state instantiation of a node so that x_{ik} denotes the kth instantiation of the ith node. Finally, let c_i be the number of states of the ith node and q_i be the number of parent states for the parent of the ith node. This means that the joint probability for a network model M is:

$$p(M) = \prod_{i=1}^{I} \prod_{j=1}^{q_i} \prod_{k=1}^{c_i} p(x_{ik}|\pi_{ij}) \tag{4.6}$$

Let us also use the notation used by Iverson [7] which uses brackets to enclose a true-false statement [7], so that if the result in the brackets is true, the result is 1 and 0 if the statement is false. As a note, when the result is 0, the term is "very strongly zero." Using this notation, we may use the expression:

$$p(M) = \prod_{i=1}^{I} \prod_{j=1}^{q_i} \prod_{k=1}^{c_i} |p(x_{ik}|\pi_{ij}) - [p(x_{ik}|\pi_{ij}) < 0.5]| \tag{4.7}$$

to quantify our confidence in the a model.

Let us take the example further and rather than having one parent node, let us consider two parent nodes each with a single possible node state. Because, naïve Bayes dictates that for a total of m parents $b_k \in B$, where B is the set of parent nodes, the probability for child A is:

$$\sum_{l=1}^{m} \frac{p(A|b_l)}{p(b_l)} \tag{4.8}$$

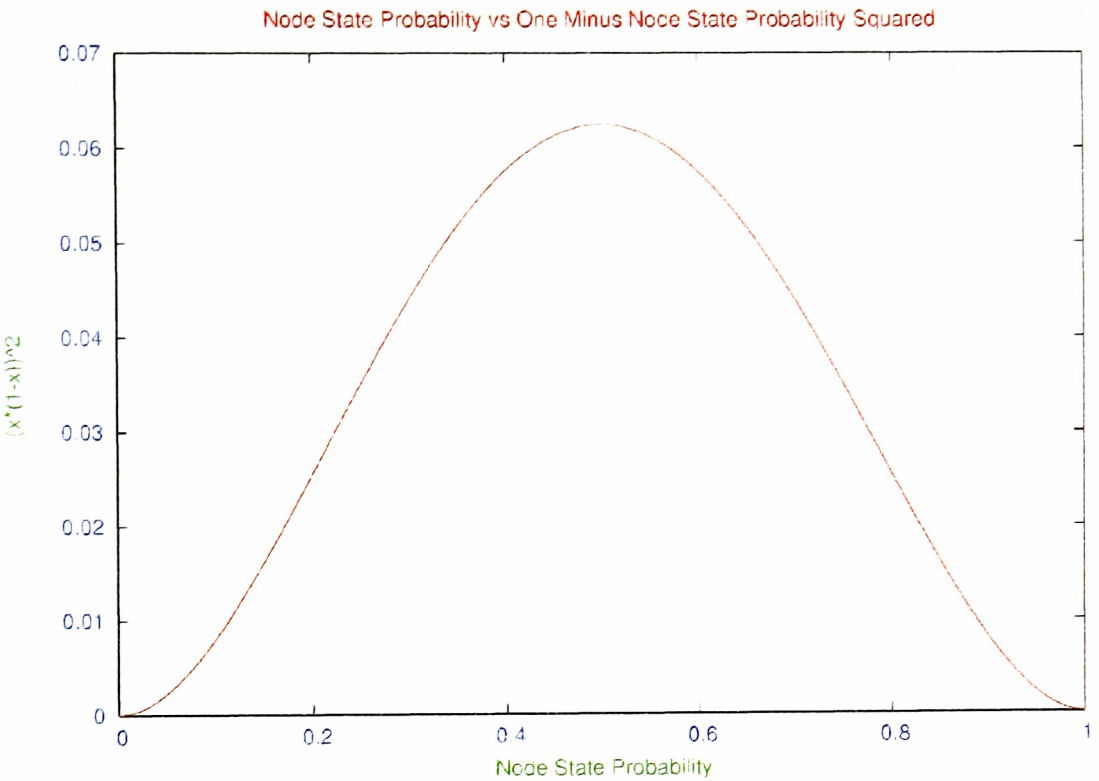


Figure 4.3: The naïve joint probability distribution function of a two-state child with two one-state parents.

The joint probability distribution for the naïve case becomes $((x(1-x))^2$ which is shown in Figure 4.3.

There are two distinguishing characteristics which differentiate the joint probability distributions of a child node in the non-naïve case (Figure 4.2) and the distribution for the naïve case (Figure 4.3) in the multi-parent case. First, the variance of the naïve case distribution function is smaller. This means that a greater percentage of the domain is

at a value with less magnitude. In structure finding, this translates to a greater difficulty in distinguishing between local structures. Secondly, the function power is much less in the multi-parent case. In structure finding, this means that although parents may do an excellent job in providing a context for its child, as more parents are added, their ability to decorrelate the states of a child will diminish. Although this may at first seem like a plausible technique for limiting structural complexity, empirical evidence suggests that in structure finding, these parent-child relationships may be misinterpreted child-parent relationships yielding edges that are oriented in the wrong direction which are the least tolerable of incorrectly derived edges. A comparison between the naïve and non-naïve joint probability distribution functions is shown in Figure 4.4.

4.2 Minimum Description Length

The Minimum Description Length (MDL) fitness function is based on Rissanen's Minimum Description Length Principle which is a well studied formalism in learning theory [9]. The MDL principle is based on the idea that the best model of a collection of features is the one that minimizes the sum of the length of the encoding of the model plus the length of the encoding of the data given the model.

As an example, let there be a set of features which are mapped to a set of n points on a plane. Let each value be specified by a pair of real coordinates with fixed preci-

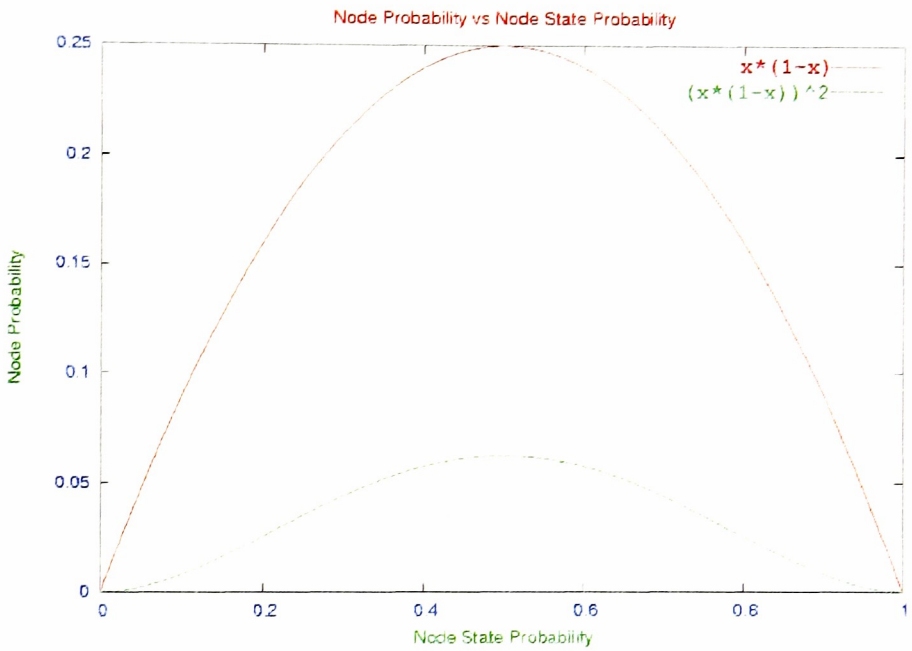


Figure 4.4: The naïve and non-naïve joint probability distribution functions of a two state child with one one-state parent.

sion $(x_1, y_1), ..., (x_n, y_n)$. Suppose then that we want to find the polynomial function (our model) that fits this set of feature values. An n degree polynomial function passing through each point would require $n + 1$ values to specify the coefficients of that function giving us the length of the encoding of the model. To store the data given the model, n x-coordinates at the given precision would be required, but the y-coordinates would not need to be stored since they can be derived from the data and the model. Hence, the sum of the description lengths would be n x-coordinates plus $n + 1$ polynomial function coefficients for a total of $2n + 1$ times the number of bits needed to store the coefficient of x-coordinate at the given precision.

Let us reduce the order polynomial function to order k. Now, we need $k + 1$ values to store coordinates. Because we are no longer guaranteeing that the y-coordinates will be precisely correct, some error δ, the distance between the y-coordinate value generated by the polynomial function and the actual y-coordinate. Now these error terms $(\delta_1, ..., \delta_k)$ must be stored along with the x-coordinates and the k polynomial coefficients. If these error terms are sufficiently small, it is possible that fewer bits would be needed to store the k order polynomial, the x-coordinates and the error terms than storing the n order polynomial and the x-coordinates. Ergo, there may be some polynomial of degree $k \leq n$ that yields a minimum description length.

To apply the MDL principle to BBN structure fitness scoring two encodings are made, the first is the encoding of the network. This corresponds to the complexity of the network.

A more densely connected network will require a larger number of bits to encode. The second encoding is that of the database given the network.

4.2.1 Encoding the Model

To represent a BBN structure, both a list of the parents of each node along with the set of conditional probabilities are necessary and sufficient. Suppose that there are n nodes for a given network. For a given node with k parents, $k \log_2(n)$ bits are required to list the node's parents. To represent the conditional probabilities, the encoding length is the product of the number of bits required to store the numerical value of each conditional probability and the total number of conditional probabilities that are required. In a BBN, a conditional probability is needed for every distinct instantiation of the parent nodes and the child node. It is important to note however, that one of these conditional probabilities per unique parent node instantiation need not be encoded, since the sum of the child node-state probabilities must sum up to one, given a set of parent node-state instantiations. Using this scheme, the total description length for a network is [18]:

$$\sum_{i=1}^{n} k_i \log_2(n) + \frac{\log M}{2} \left(\sum_{i=1}^{n} (s_i - 1) \prod_{j \in pa(i)} s_j \right) \tag{4.9}$$

where there are n nodes and for node i, s_i is the number of states it can take on. M is the number of events in the database. Using the central limit theorem, each feature has a variance of $\sqrt{M}$. Thus, for each feature in the network, the number of bits required to

encode it is [18]:

$$\frac{\log M}{2} \tag{4.10}$$

The equation clearly show that highly connected networks will require longer encodings. As the number of parents for a given node gets larger, the list of conditional probabilities that needs to be stored for that node also increases. Also, networks in which the nodes have a larger number of values will have parents with a larger number of values, which will require longer encoding lengths. Therefore, MDL tends to favor networks in which the nodes have a smaller number of parents and also networks in which nodes taking on a larger number of states are not parents of nodes that also take a larger number of states.

4.2.2 Encoding the Data Given the Model

The task of encoding the data given the model is to learn the joint distribution of a set of nodes with each node having a set of states to which it can be instantiated. This is accomplished by encoding the data as a binary string and using this encoding length as a scoring metric (as with encoding the network). It is well known that for character codes, we can minimize the length of the final binary string by taking into account the frequency of the occurrence. Huffman encoding is the optimal algorithm for generating minimum length character encoding. This algorithm assigns the most frequently occurring values into shorter codes. Thus, Huffman's algorithm requires as input the frequency of occurrence of of each event in the database. For a database with N events and underlying event i

occurring with probability p_i, the encoding length of the database will be approximately:

$$-N \sum_{i=1}^{N} p_i \log_2(p_i) \tag{4.11}$$

when summing over all events.

The problem is that we do not have the probabilities of the underlying events p_i. We do however, have an approximation of this value from the database $\hat{p}_i$. Also, the BBN acts as a model of the underlying distribution and it also assigns a probability q_i to each event. In general, $p_i \neq \hat{p}_i \neq q_i$ and the learning scheme cannot guarantee that it will construct a perfectly accurate network. Still, the goal is to have q_i approach p_i, and the closer these terms become, the more accurate the model becomes.

A candidate BBN structure is intended as a "guess" model of the underlying distribution. Thus, given the probabilities q_i determined by the network and a best guess of the true values $\hat{p}_i$, a Huffman code is designed using these probabilities. Each event is assigned a codeword of length approximately $-\log_2(q_i)$ with probability of occurrence p_i yielding a BBN encoding length of:

$$-N \sum_{i=1}^{N} \hat{p}_i \log_2(q_i) \tag{4.12}$$

for N events in the database.

A question arises as to how this encoding length compares to the encoding length if we had the underlying probabilities p_i. A theorem due to Gibbs gives the answer. If we let p_i

and q_i where $i = 1, ..., N$ be non-negative real numbers that sum to 1, then:

$$-\sum_{i=1}^{N} p_i \log_2(p_i) \leq -\sum_{i=1}^{N} p_i \log_2(q_i) \tag{4.13}$$

with equality holding if and only if $\forall i,\ p_i = q_i$. In this summation, $0 \log_2(0)$ is taken to be 0. This theorem shows that the encoding length using the estimated probabilities will yield a longer encoding than if the true probabilities are used. It also says that the true probabilities yield the minimum encoding length.

From a more intuitive standpoint, we see that Equation 4.12 takes the negative of the logarithm of the joint probability of a BBN given node instantiations i in the database and is weighted by the number of occurrences of i in the database. This means that this scoring technique will favor networks with more certainty in the most probable cases, yielding greater precision for the most probable events.

4.2.3 Analysis of MDL

As stated above, the MDL fitness function gives a more optimal score to networks where the length of the encoding of the model plus the length of the encoding of the data given the model, both of which are measured in bits, is minimized. MDL in equation form looks like:

$$\sum_{i=1}^{n} k_i \log_2(n) + \frac{\log M}{2} \left(\sum_{i=1}^{n} (s_i - 1) \prod_{j \in pa(i)} s_j \right) - N \sum_{i=1}^{N} \hat{p}_i \log_2(q_i) \tag{4.14}$$

or more simply:

$$L_{network} + L_{data} \tag{4.15}$$

where $L_{network}$ is the encoding length of the network and L_{data} is the encoding length of the data given the network.

As an efficient space searching heuristic, MDL's strength lies in its simplicity. First, when finding $L_{network}$, MDL is not concerned with the actual node-state probabilities. To find the encoding length for a given node, only the number of states of the given node and its parent are needed. The network encoding length is not concerned with the actual values for any given node instantiation. Second, when finding L_{data}, MDL is only concerned with the joint probabilities for unique events in the database, and not the joint probabilities for every possible network instantiation.

From a computational complexity standpoint, we see that to calculate the the MDL score for a given network with n nodes given a database, we must find the conditional probability of the child given the parents. The complexity will be

$$O(n \cdot 2r) \tag{4.16}$$

where r is the cost of finding the probability of intersection of a set of nodes with a given instantiation since each node is always iterated through when finding the network encoding length. On the other hand, when finding the data encoding length, all network nodes must be instantiated for each unique database event e_u, and for each of these unique event, we

must find the modified joint probability of the entire network meaning that each time MDL is calculated for a network a computational cost of

$$O(e_u \cdot n \cdot 2r) \tag{4.17}$$

must be paid.

We see that, in general, MDL suffers from three major problems. First, as stated before, MDL will favor structures where nodes taking on a larger number of states are not parents of nodes that also take a larger number of states. Although this functions as a benefit in the area of space and time complexity, it is very possible that this will preclude or, at least discourage, structures that may be correct simply because of the number of states in the nodes. This problem could easily be solved by removing the term in the network encoding which multiplies the number of states of a given node by the number of states of each of its parents. The second problem incurred by MDL comes from the fact that the data encoding looks at database events where an event is a node instantiation for all nodes in a network. Because the number of possible node instantiations may be extremely large, the probability of any given network node instantiation is going to be extremely small for all but the largest databases, and so a common local structure in the network may not be differentiated or favored that much more than a much less common structure. Finally, the two encoding expressions are not normalized with respect to each other. To compensate for this, a scaling factor may be added to either of the terms, but it is still not clear as to how a scaling value should be be chosen. A better approach may be to look at the probability of

local structures when determining which node instantiations should be favored.

4.3 The Database Probability Given a Model

Whereas MDL uses indirect information about the network structure to give it a score, the fitness function calculating the probability of the database given a model (DGM), which is taken from a method proposed by Ramoni and Sebastiani [17], takes a more direct approach.

Given a database D of events, we wish to construct a model M of conditional dependencies from features in a database. Taking a Bayesian approach, we can take $p(M)$ to be the prior belief in a particular model. The information in the database can then be used to compute the posterior probability of M, given the data:

$$p(M|D) = \frac{p(M, D)}{p(D)} \tag{4.18}$$

In comparing two models M_1 and M_2, we can use Bayes factor:

$$\frac{p(M_1, D)}{p(M_2, D)} \tag{4.19}$$

thereby eliminating the need to find the database probability $p(D)$.

Let $x \in X$ be the set of nodes in a network with x_i denoting the ith node, and let k denote the state instantiation of a node so that x_{ik} denotes the kth instantiation of the ith node. Next, let $n(x_{ik}|\pi_{ij}), i = 1..., I; j = 1, ..., q_i; k = 1, ..., c_i$ be the frequency of cases in

the database with $x_{ik}|\pi_{ij}$, so that

$$n(\pi_{ij}) = \sum_{k=1}^{c_i} n(x_{ik}|\pi_{ij}) \tag{4.20}$$

is the frequency of cases with π_{ij}. The probability of the intersection of a model and database can then be given as [17]:

$$p(M, D) = p(M) \prod_{i=1}^{I} \prod_{j=1}^{q_i} \frac{(c_i - 1)! \prod_{k=1}^{c_i} n(x_{ik}|\pi_{ij})!}{(c_i + n(\pi_{ij}) - 1)!} \tag{4.21}$$

It may be noted that:

$$p(M, D) = p(M) \prod_{i=1}^{I} \prod_{j=1}^{q_i} \frac{(c_i + n(\pi_{ij}) - 1)!}{(c_i - 1)! \prod_{k=1}^{c_i} n(x_{ik}|\pi_{ij})!} \tag{4.22}$$

is the number of ways in which the frequencies could have been ordered in the database, and $p(M, D)$ is proportional to the inverse of the number of possible databases that could give the observed count.

If the $p(M)$ term is eliminated from Equation 4.21, then we are left with:

$$p(D|M) = \prod_{i=1}^{I} \prod_{j=1}^{q_i} \frac{(c_i - 1)! \prod_{k=1}^{c_i} n(x_{ik}|\pi_{ij})!}{(c_i + n(\pi_{ij}) - 1)!} \tag{4.23}$$

This equation quantifies our confidence in a set of database statistics given a structural model and this is the function that we will use as a fitness function. Heuristic space searching continues while adding edges continues to increase $p(D|M)$ and stops when the probability no longer increases.

To further illustrate the utility of this fitness function, let us use the example from Section 4.1 where we are looking at a single state single parent with two state child. In this

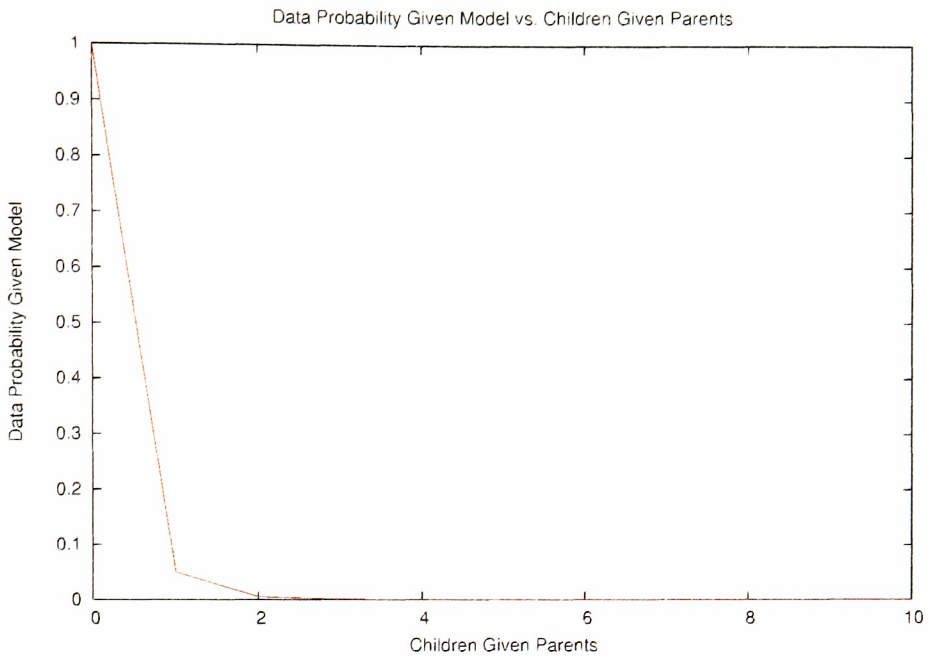


Figure 4.5: The contribution of an edge to an example network with 20 parents

case, the contribution the node and its parent will be:

$$g(x_1, p_1) = \frac{n(x_{1,1}|\pi_{1,1})!(n(\pi_{1,1}) - n(x_{1,1}|n(x_{1,1}|\pi_{1,1}))!}{n(\pi_{1,1})!} \tag{4.24}$$

Put more generally, the local contribution of node x_i and its parents π_i to the probability the database D given the model M can be measured by:

$$g(x_i, \pi_i) = \prod_{j=1}^{q_i} \frac{(c_i - 1)! \prod_{k=1}^{c_i} n(x_{ik}|\pi_{ij})!}{(c_i + n(\pi_{ij}) - 1)!} \tag{4.25}$$

Figures 4.5 and 4.6 give a graph representation of Equation 4.24 for the first half of the symmetric function when the parent count in a database is 20 and 100 respectively. From the graphs we see that as the number of parents increases the slope showing the transition

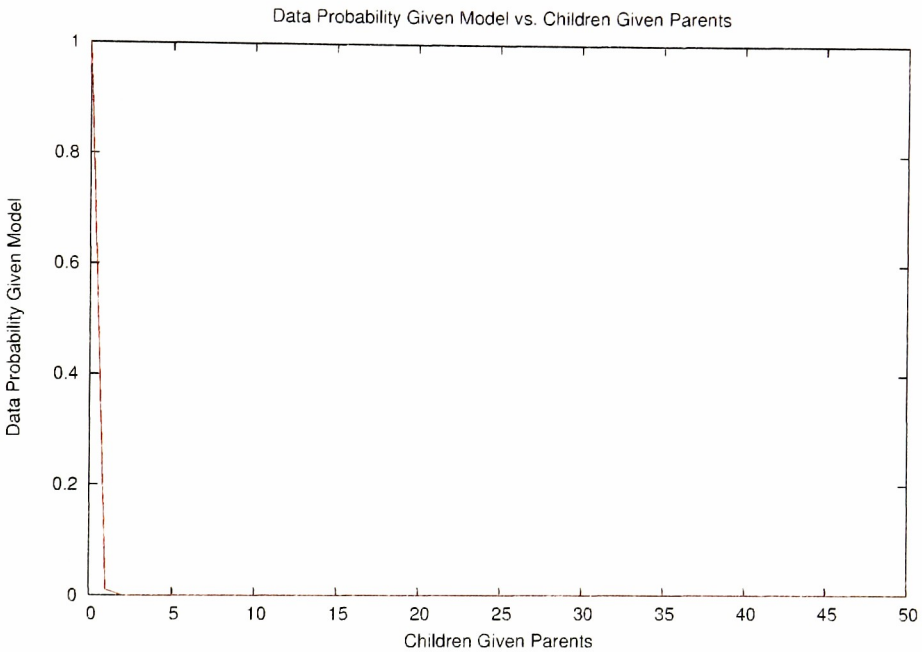


Figure 4.6: The contribution of an edge to an example network with 100 parents

from a high level of confidence to a low level of confidence where the number of children in the database given a parent is large or small increase and decreases respectively as a function of the the number of children given a parent. This means that when there are fewer parents, the child given parent confidence is more likely to be somewhere between the the extremes corresponding to less confidence in the connection. On the other hand, when the parent count is higher in the database, we are more likely to have either a high or low confidence that the candidate child node is or is not in fact a child node of the parent.

For a computational complexity standpoint, we see that to calculate DGM for a network, we must take the product of $g(x_i, p_i)$ for each node that has parents. For each of these calculations, the calculation of of the factorial of the number of parents, the intersection count in the database between the node instantiation and the parent instantiation and the factorial of the intersection count must be found for each possible instantiation of parents. Taking these value into account we get a complexity of

$$O(n \cdot s_p \cdot s_n \cdot f^2 \cdot r^2) \tag{4.26}$$

where s_p is the number of parent node instantiations for a node in a network , s_n is the number of states of a node in a network, f is the cost incurred by computing a factorial and r is the cost of computing the intersection of a node given an instantiation and a set of parents given an instantiation.

Because it directly quantifies how well a database is modelled by a BBN, the DGM scoring heuristic is accurate. However, it does suffer from two problems. First, DGM tries

to model a database with a BBN directly. As a result, DGM will continue to add edges as long as doing so creates a BBN that better models the database. This means that DGM will over-fit the network to the database. Fortunately, this problem can easily be remedied by applying a threshold to $g(x_i, p_i)$. It is important to note, that by doing this, we are not simply defining an arbitrary threshold. A threshold value actually quantifies how much statistical expectation is required to justify an explanation or causation in a particular BBN model.

Unfortunately, the second problem is not as easily remedied. It may be noted that the terms in the DGM equations involve factorial calculations, and as a result, certain terms increase very quickly. This problem lends itself to not only a large number of operations to find the DGM score, but also a problem of precision in the implementation of this function. In this thesis, implementation required a multiple precision library to cope with this problem. However, it would also be possible to take the logarithm of the DGM equations which would not only shorten the score representations, but also decrease the slower multiplication and division operators at the cost of needing to find logarithms of the terms.

It may also be noted that since the DGM score is the product of local structure scores that Equation 4.25 could be used to create a network, and in fact, this equation can be used in greedy structure finding. Iterations consist of temporarily adding a single edge for all possible single edge additions and, the edge whose local contribution score is the highest is added to the network and the process is repeated. This method is much more efficient than

finding the global structure, but remains just as accurate.

In realizing that DGM takes the product of local structure scores, a technique does present itself that allows us to quantify the DGM difference between two arbitrary networks easily, without needing to take the global structure score. By doing a subtraction of the binary representation of two BBNs to find the DGM difference between, we see that edge descriptors, whose values are not zero correspond to differences in the structures. By creating BBN subgraphs based only on the differences between the structures and then using Equation 4.23 on these subgraphs, only the differences in local structures are compared and the redundancy of commonality between the original structures is eliminated. This technique would increase the efficiency in comparing BBNs not only in this thesis but, also when other techniques such as Particle Swarm Optimization or pure Genetic Algorithms are employed.

We see that fitness functions within the heuristic search framework directly quantify how well knowledge database features are modelled in a BBN. However, it may also be noted that variation required by the framework will require that fitness functions be applied to many candidate structures per generation. As a result, the heuristic search framework is not as efficient as the asymptotically correct approach. In the next chapter, we combine the approaches, yielding a model selection technique that is more precise than a pure asymptotically correct approach while remaining more efficient than a pure heuristic search approach.

Chapter 5

Proposed Approach

The asymptotically correct structure learning algorithm is efficient. It can search a relatively large space of possible BBN structures in a relatively short amount of time. However, it is not always the most precise. In the case of the ALARM network, for the asymptotically correct structure learning algorithm described, eight edges were not derived, two extra edges were added, and up to one is incorrectly oriented when using a database of 10,000 events. While this by no means rules out the feasibility of this method, which continues to be the least computationally complex, it does mean that when a high degree of precision is demanded, this may not be the best algorithm to use by itself.

Heuristic search methods have the ability to be accurate. They have the ability to exploit small areas of a BBN structure search space. When tested using the ALARM network, space searching using MDL yielded a structure that resembled the original network.

Two arcs were oriented incorrectly and three extra edges were added [9]. DGM is the most accurate scoring function proposed by Ramoni and Sebastiani [17] when proposing Bound and Collapse to perform model selection on a database with incomplete data. Using the Bayesware Discoverer software package [1], which uses this scoring scheme, on the ALARM network, one edge was not found and one extra edge was added. As mentioned before though, this precision comes at a cost of computational complexity.

This thesis uses a combined approach, which allows for network precision as well as the ability to construct a network quickly. By starting with the asymptotically correct structure learning for efficient search space exploration, a candidate network that is close to the underlying structure is found quickly. Then, a heuristic search is done to refine the network. In this way, a network is derived with more computational efficiency than if a pure heuristic search is used, and at the same time, a more accurate structure is derived than if a pure asymptotically correct structure learning algorithms is used.

As was shown before, the computational cost of calculating DGM and MDL was $O(n \cdot s_p \cdot s_n \cdot f^2 \cdot r^2))$ and $O(e_u \cdot n \cdot 2r)$ respectively. Strictly in terms of the number of nodes, we have a computational complexity of $O(n)$ for both DGM and MDL. In the context for the described framework, we can reduce the complexity in the DGM case. Because, for each generation, any given variation of a node varies from the node from which it is derived, n calculations can be saved by finding only the difference between a variation and the candidate from which it was varied. Therefore, in the context of this structure searching

framework, the computational complexity of DGM can be taken as $O(1)$ in terms of the number of nodes in the network.

It is important to note that the above calculations are for a single network score only. In the described framework, this calculation must be done for each of $n(n-1)$ candidates for up to $\frac{n(n-1)}{2}$ generations. Taking this into account, we see that the algorithm complexity for the heuristic search is, at best, $O(n^4)$, but may be $O(n^5)$ depending on the scoring function.

The computational cost of the heuristic search is large, and if possible, we would like to avoid needing to pay for it. However, as noted before, this search lends itself to extremely precise networks. In looking at the cost, we see that a large portion is paid by the number of generations that must be iterated through, and if we could reduce the number of iterations needed to reach a final structure, we could reduce the overall complexity.

Suppose that we are able to start with an initial structure that will reduce the number of generations needed to reach the final structure. Let $g_{i,f}$ be the number of generations to get from our initial given structure to a final structure. We see that:

$$g_{i,f} = n(e_e) + n(e_m) \tag{5.1}$$

where $n(e_e)$ and $n(e_m)$ is the number of extra edges and missing edges between our initial given network and our final network respectively. Taking into account $g_{i,f}$ we see that the computational complexity of our new scheme for DGM is:

$$O(g_{i,f} \cdot n^2) \tag{5.2}$$

and, as long as $g_{i,f}$ is less than the number of edges we need to add to a fully disconnected network to get to a final structure, we have reduced the cost space searching cost.

The problem remains still, of the getting an initial network structure which will, in turn, dictate $g_{i,f}$. We see that, in general, our initial guess algorithm must have a computational complexity $O(Q)$ such that:

$$O(g_{i,f} \cdot n^2 + Q) < O(n^4) \tag{5.3}$$

In this way, we ensure that the algorithm used for our initial guess combined with our heuristic search remains more computationally efficient that the heuristic search.

With it computational complexity of $O(n^2)$, we see that the asymptotically correct information theory approach fits this criterion. Filling this information into Equation 5.3, we get:

$$O(n^2(1 + g_{i,f})) < O(n^4) \tag{5.4}$$

Thus the information theory structure finding algorithm appears to be an excellent candidate for finding an initial structure which a heuristic search can then refine, and this is the approach taken in this thesis to deriving a statistically precise model while reducing the cost normally associated with algorithm that attain this precision.

It may have been noted that it was not proven that either of the presented fitness functions are convex on either the database in the case of DGM or the network in the case of MDL, and therefore, it has not been proven that a global optimum will be found regardless

of the initial starting point when a heuristic search is performed. In fact, in the case of MDL, we see that it is unlikely that a convex function is defined and as a result there are likely local optimums for a given BBN.

In the case of DGM, we see that proving convexity is a matter of proving that $g(x_i, \pi_i)$ is itself a convex function. If this is the case, then we may define the optimal structure as having a maximum score S_{op} such that:

$$S_{op} = \prod_{i=1}^{I} g_o(x_i, \pi_i) \tag{5.5}$$

where $g_o()$ is the optimal $g()$ value. As a proof, we start with the fact that:

$$S_{op} > \prod_{i=1}^{I} g(x_i, \pi_i) \tag{5.6}$$

where the $g(x_i, \pi_i)$ terms describe the $g()$ values of a sub-optimal structure. We see that if some:

$$g_o(x_i, \pi_i) < g(x_i, \pi_i) \tag{5.7}$$

then it would be possible to construct a network with a higher score than the optimal. However, by definition, the optimal structure yields the highest DGM score and we have a contradiction.

The matter of proving $g(x_i, \pi_i)$ is a convex function on a database is much more difficult problem because the DGM equation is not in closed form. However, the application of DGM as shown in Chapter 6 offers some evidence that it may be convex in the case of the ALARM network.

Along with testing the proposed two-step algorithm, a comparison is provided between the data given model probability verses the minimum description length in the context of heuristic scoring. We see that because it utilizes database events as opposed to feature probabilities, MDLs computational complexity is tied to the unique events in the database and as the database grows MDL becomes more expensive to compute than DGM. Also, it has not been shown that database event probabilities are a valid features to derive a network structure. On the other hand, because it looks at the probabilities of node states in the database, DGM has the potential to be difficult to compute because of the magnitude of some of the terms. Performance and results for both DGM and MDL in both non-näive and näive BBNs are provided.

Chapter 6

Results

The experiments in this thesis started by constructing a BBN starting with a fully disconnected graph. MDL and DGM heuristic searches were then performed for both naïve and non-naïve BBNs. Next, a BBN was constructed using the asymptotically correct structure learning algorithm based on the information theory approach as it was implemented in PowerConstructor. For this part of the experiment, the mutual information threshold value was left at its default value. Increasing this value would have lead to a more sparsely connected network with a higher degree of confidence in the correctness of both the edges and their orientation and decreasing this value would have yielded a more connected network. The final step was to use the heuristic search in an attempt to refine the network. Both the MDL and DGM scoring functions were tried in this step.

Model Selection	Correct Edges	Missing Edges	Extra Edges	Wrong Oriented
MDL, N, FD, C	17	26	10	3
MDL, N, FD, B	13	29	5	3
MDL, N, FD, C	12	34	5	2
MDL, N, FD, B	14	32	4	3
Info. Theory	38	8	2	1
DGM, FD/ITI , C/B	45	1	1	0

Table 6.1: The tabular representation of the results

6.1 Tabular Representation

To provide a condensed representation of the results, Table 6.1 is provided. The model selection technique is formatted to indicate the algorithm used, whether a naïve or non-naïve network was used, the initial network used, and which database was used. N indicates naïve NN indicates non-naïve, FD indicates a fully disconnected initial network ITI indicates an information theory initial structure, C indicates Cheng's database and B indicates the BNT database. Once again, we see that DGM converges to the most precise structure regardless of either the database or the starting point. MDL on the other hand performs poorly compared to not only DGM, but also the information theory approach. Also, MDL is unable to refine a network starting with the structure derived the information theory approach.

6.2 Fully Disconnected Initial Network

6.2.1 MDL with Naïve BBN

Figures 6.1 and 6.2 show the results of attempting to create the ALARM network from a fully disconnected naïve BBN from Cheng's database [3] and a database created by BNT respectively [11]. In the case of Cheng's database, 26 edges were not found, 10 extra edges were added and 17 edges were correctly derived. Of the incorrect edges, there were 3 that were incorrectly oriented. In the case of the BNT database, 29 edges were not found, 5 extra edges were added and 13 edges were correctly derived. Of the incorrect edges, there were again 3 that were incorrectly oriented. We see that although both do poorly, the former database creates more correctly connected edges than the latter. It should be noted that Cheng's database has fewer unique events than databases created by BNT and, although they may be statistically correct it did not show the diversity of events as those generated by BNT did.

6.2.2 MDL with Non-Naïve BBN

Figure 6.3 shows the results of attempting to create the ALARM network using MDL with a non-naïve Bayes model with Cheng's database. As shown, 34 edges were not found, 5 extra edges were added and 12 edges were correctly derived. Of the incorrect edges, 2 edges were incorrectly oriented.

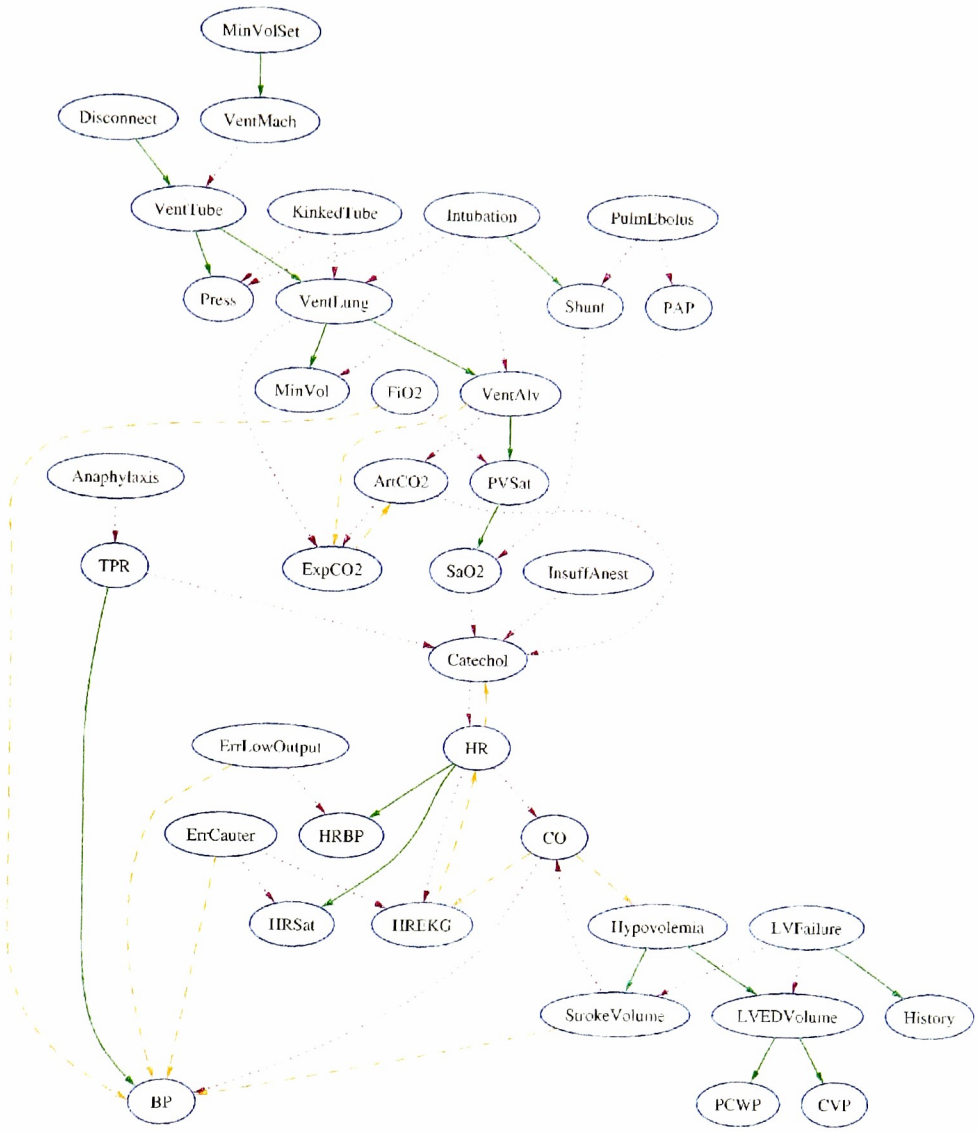


Figure 6.1: The network derived from MDL using a naïve Bayes model with Chen's database starting with a fully disconnected network. Dotted edges indicate edges that are present in the ALARM network, but were not found. Dashed lines indicate extra edges found using MDL that do not appear in the ALARM Network. Solid edges indicate correctly found edges.

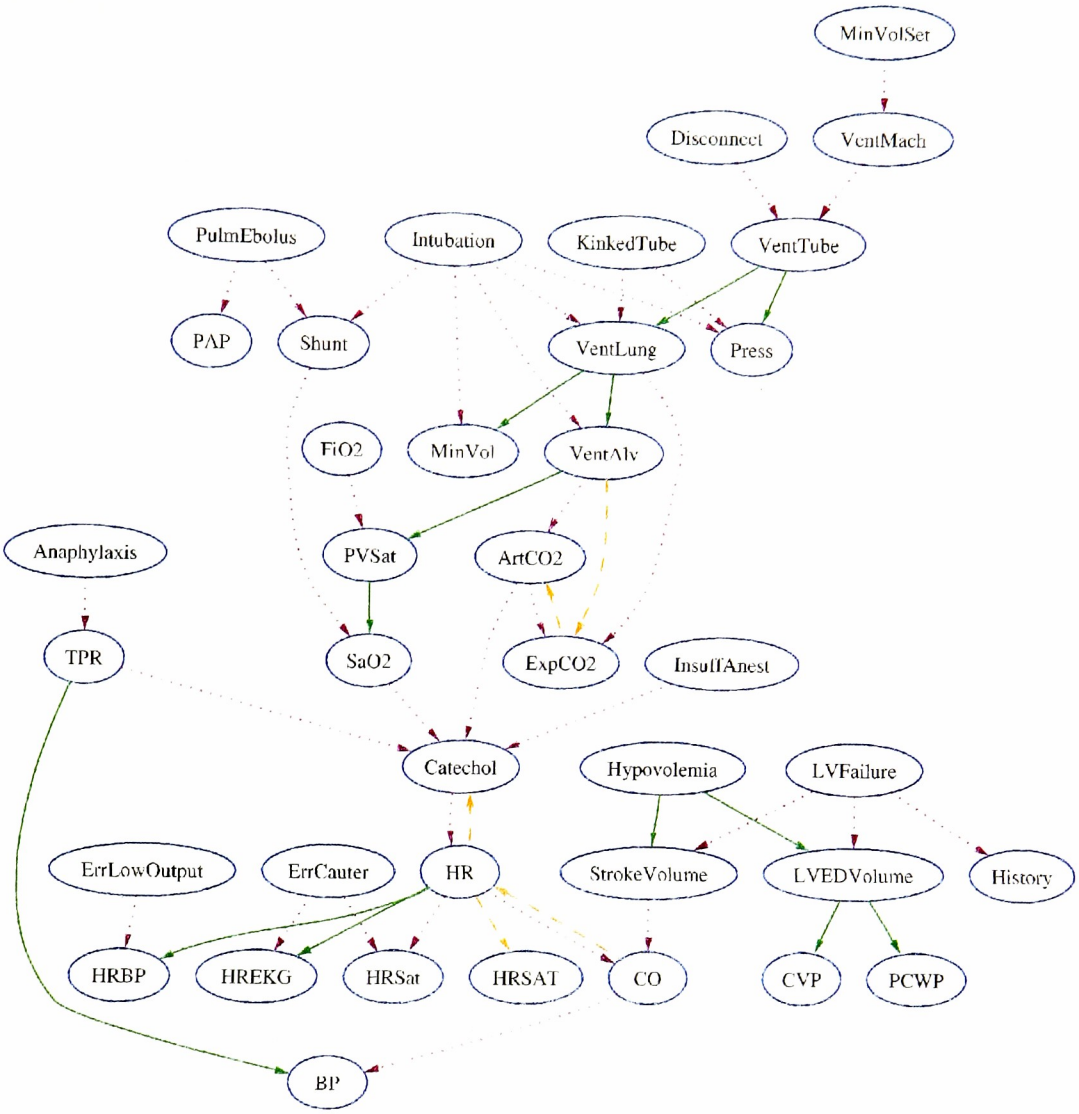


Figure 6.2: The network derived from MDL using a naïve Bayes model with a database created by BNT starting with a fully disconnected network. Dotted edges indicate edges that are present in the ALARM network, but were not found. Dashed lines indicate extra edges found using MDL that do not appear in the ALARM network. Solid edges indicate correctly found edges.

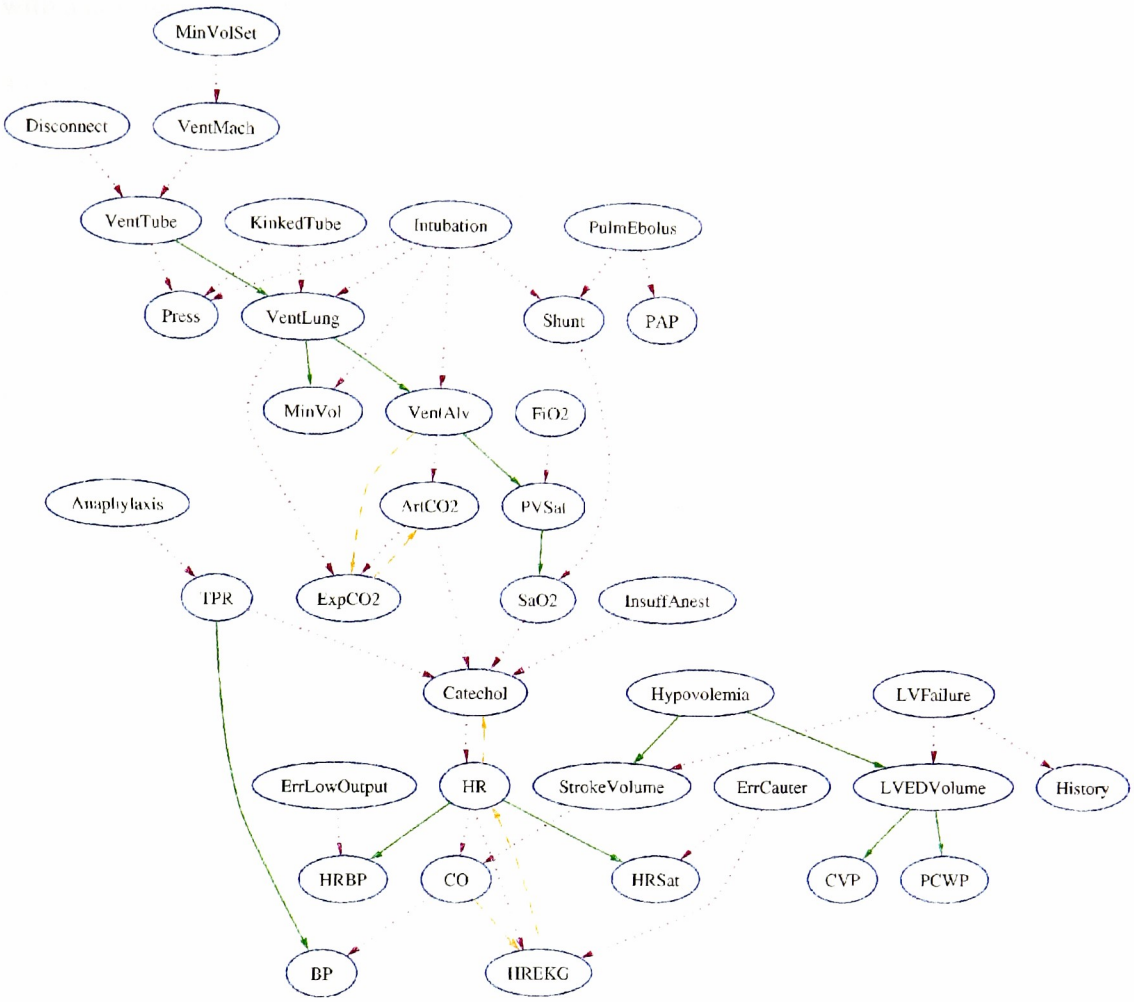


Figure 6.3: The network derived from MDL using a non-naïve Bayes model with Cheng's database. Dotted edges indicate edges that are present in the ALARM network, but were not found. Dashed lines indicate extra edges found using MDL that did not appear in the ALARM network. Solid edges indicate correctly found edges.

Figure 6.4 shows the results of attempting to create the ALARM network using MDL with a non-naïve Bayes model with the BNT database. As shown, 32 edges were not found, 4 extra edges were added and 14 were correctly derived. Of the incorrect edges, 3 edges were incorrectly oriented.

6.2.3 Data Given Model Probability

Figure 6.5 shows the results of attempting to create the ALARM network from a fully disconnected BBN using DGM with both Cheng's database and a database generated by BNT as they yielded the same structure. It is important to note this figure shows the structure derived using a threshold that minimizes the number of differences between the the derived structure and the original structure. As shown in the figure, one edge was not found, one extra edge was added and, of the incorrect edges none corresponded to wrongly oriented edges.

6.3 Refining the Information Theory Structure

6.3.1 MDL with Naïve and Non-Naïve BBN

For MDL in both the Naïve and Non-Naïve case, for both Cheng's database and a database created by BNT, the structures found by MDL begin diverging immediately.

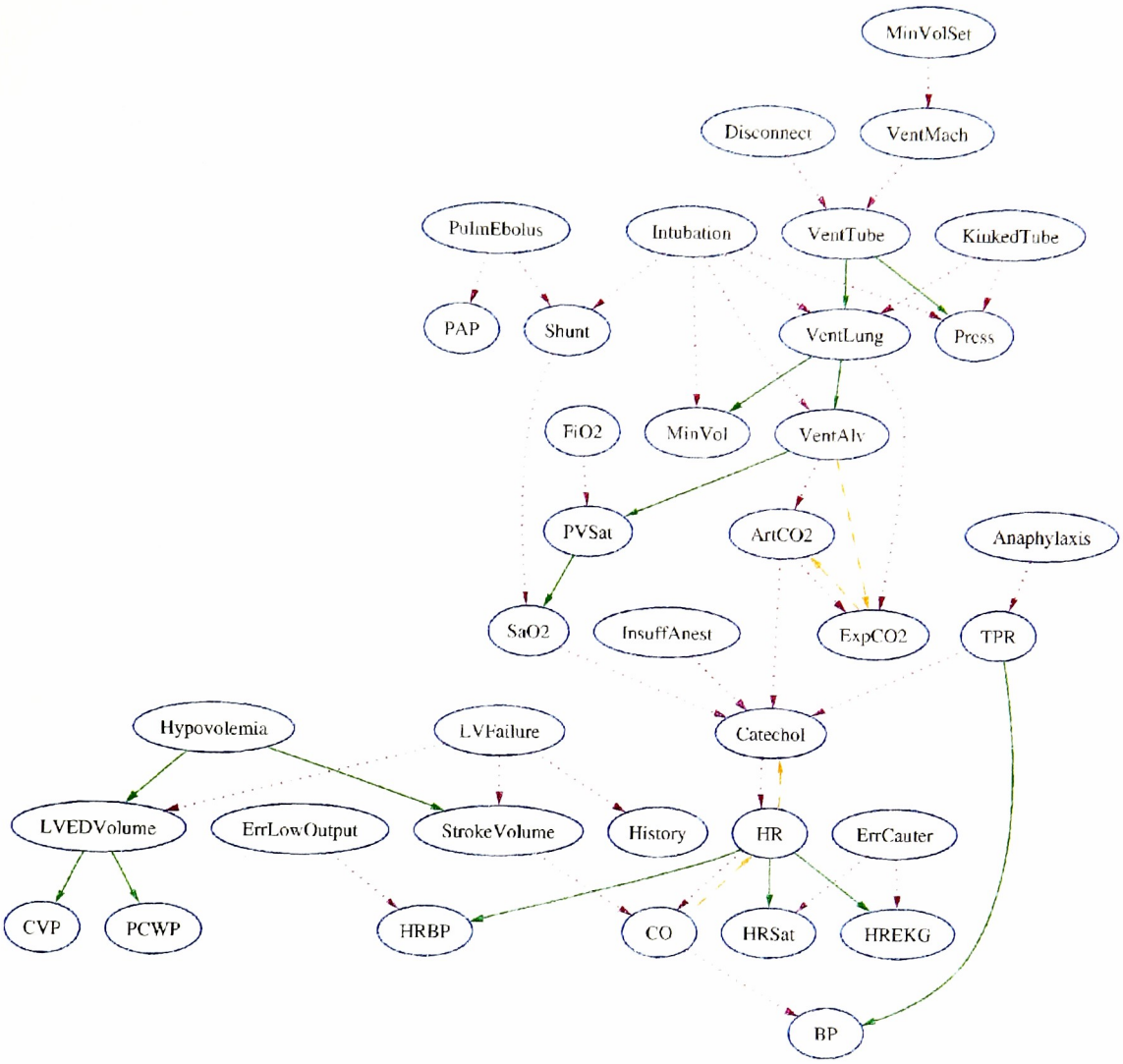


Figure 6.4: The network derived from MDL using a non-naïve Bayes model with a database created by BNT starting with a fully disconnected network. Dotted edges indicate edges that are present in the ALARM network, but were not found. Dashed lines indicate extra edges found using MDL that did not appear in the ALARM network. Solid edges indicate correctly found edges.

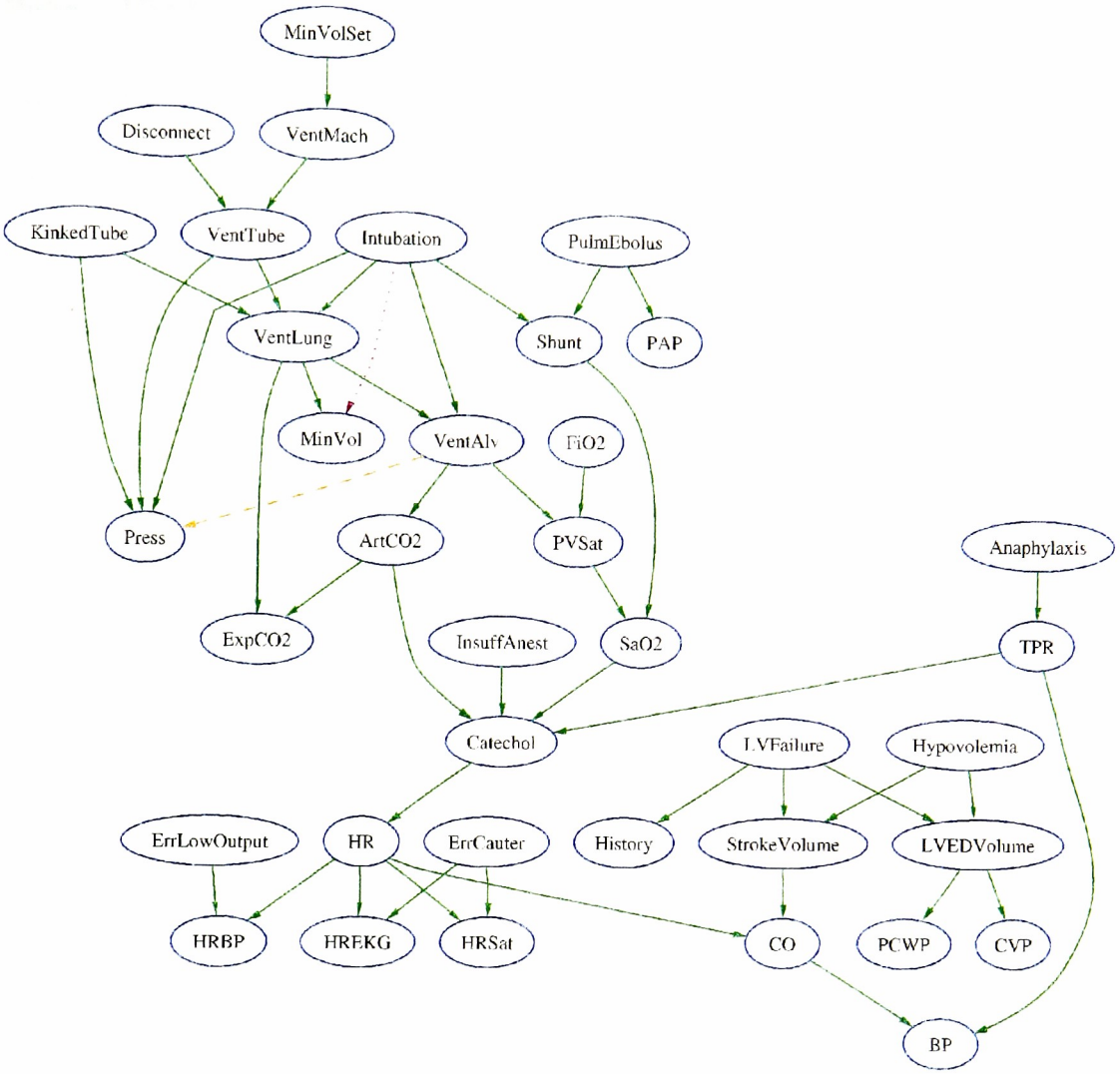


Figure 6.5: The network derived from DGM for both the Cheng's database and one created by BNT starting with a fully disconnected network. Dotted edges indicate edges that are present in the ALARM network, but were not found. Dashed lines indicate extra edges found using MDL that did not appear in the ALARM network. Solid edges indicate correctly found edges.

6.3.2 DGM

Figure 6.5 also show the results of attempting to create the ALARM network from starting with the asymptotically correct information theory BBN for both Cheng's database and one generated by BNT, as both of these databases also yielded the same structure. Once again, this figure shows the structure derived using a threshold that minimized the number of discrepancies between the original and derived structure. As shown in the figure, one edge was not found, one extra edge was added and, of the incorrect edges none corresponded to wrongly oriented edges.

This chapter has provided the raw results. We have see the results of using MDL, in both the non-naïve and naïve case, and DGM starting from both a fully disconnected network and the network derived from the asymptotically correct information theory approach. In general, DGM yields a network that is structurally more like the underlying structure than MDL. However, it is computationally more complex than MDL. It has also been shown that in the case of the ALARM network, DGM is a provides a reasonable fitness function for model refining when the initial network was derived from the asymptotically correct information theory approach. We have also seen that the asymptotically correct information theory approach is computationally simpler than the heuristic scoring approach, but does not tend to yield as precise a structure as heuristic scoring.

With these results in mind, the next chapter will provide an analysis of the model selection techniques employed. Analysis will include strengths and weaknesses as well as

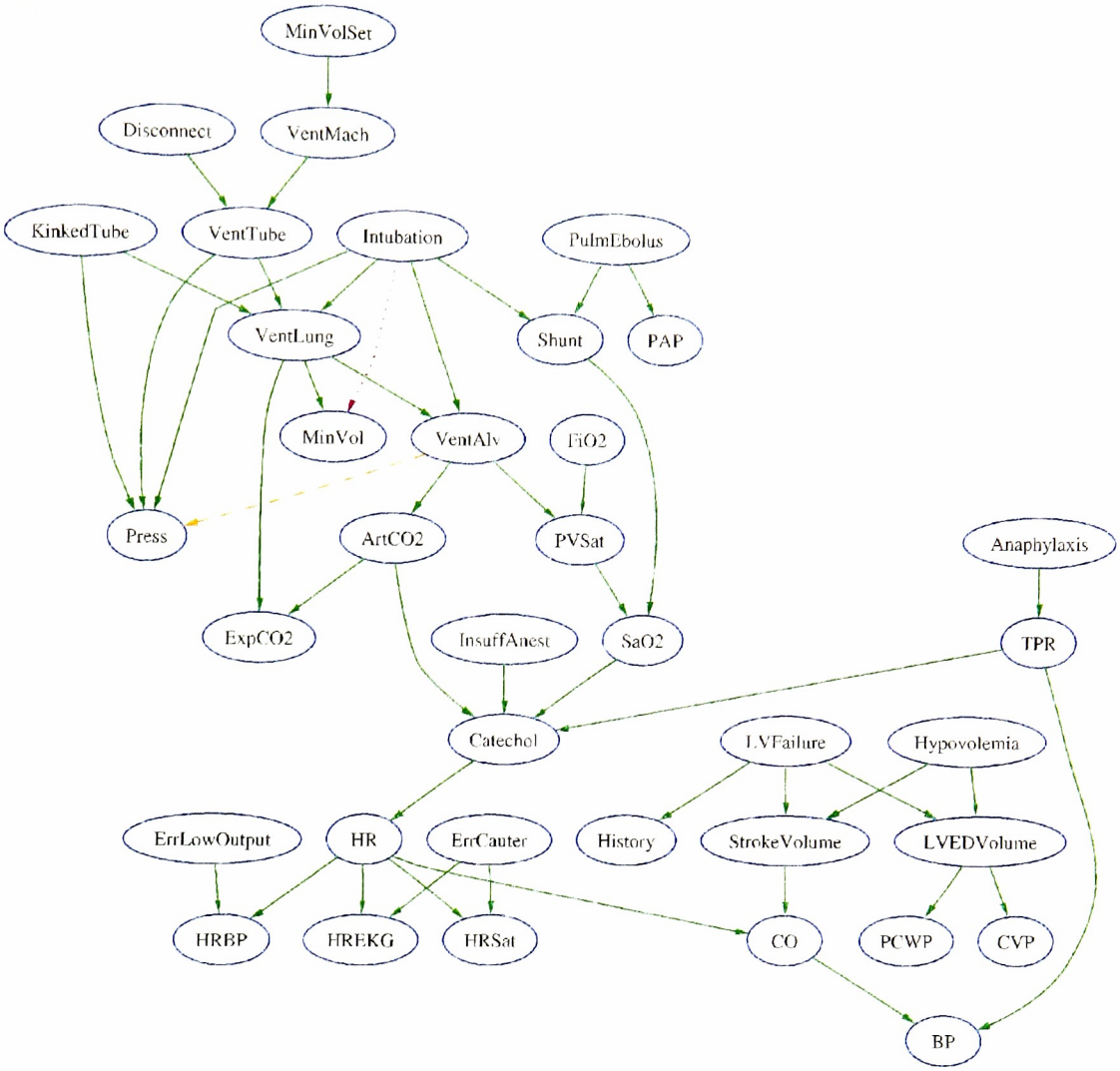


Figure 6.6: The network derived from DGM for both the Cheng's database and one created by BNT starting with a fully disconnected network. Dotted edges indicate edges that are present in the ALARM network, but were not found. Dashed lines indicate extra edges found using MDL that did not appear in the ALARM network. Solid edges indicate correctly found edges.

suggestions for both improvements of the current algorithms as well as suggestions for future work. The next chapter provides a discussion and analysis of the frameworks and the fitness functions.

Chapter 7

Discussion

The results show that by using the asymptotically correct information theory approach to BBN structure learning for search space exploration followed by refining the network using a heuristic scoring approach for search space exploitation, it is possible to derive a BBN that is more accurate and while incurring less computational complexity than using any single method described. As shown before, the information theory approach incurs a computational complexity of $O(n^2)$. The heuristic scoring approach incurs a complexity of $O(n^3)$ in term of node complexity. However, by first using the information theory approach we are offsetting some of the complexity of using the heuristic scoring approach with the computationally simpler information theory approach and, as a result the number of generations g that the heuristic search must iterate through is reduced to $O(g \cdot n)$ such that $O(n^2(1 + g_{i,f})) < O(n^4)$ This calculation will generally be simpler in terms of complexity

than than the original and, we see that as long as the asymptotically correct algorithm adds more correct edges than incorrect edges, the heuristic search has a smaller space to search and converges more quickly than if only a heuristic search has been done. In practice we see that for the ALARM network $g_{i,f}$ is 11 with DGM when refining the network. Since the number of generations needed to arrive at the same structure is 34 generations less than if we had started from a fully disconnected network, we have considerably reduced the time needed to perform the heuristic search by reducing the number of generations we must iterate through. At the same time, by using the heuristic search, which has been shown to be more precise than the asymptotically correct algorithm, we are selecting a model that facilitates more accurate inference.

The results also show that although MDL has been shown as a viable method for structure learning, in the context of both starting from a fully disconnected network as well as refining a network by using it as a scoring heuristic as it was described, it does not offer a viable solution, and it has been shown that a DGM structure will yield a more precise network while the asymptotically correct information theory structure learning algorithm will produce a more precise network with less computational complexity. The reason for MDL relative ineffectiveness is threefold. First, MDL has a bias against connecting nodes with a large number of states to other nodes with large numbers of states. Although this bias keeps the resulting networks simple, it may preclude the construction of the best structure due to the number of states of certain nodes. The second problem is that MDL weighs the

certainty of the node instantiations for an entire network by the probability of that global event in the database. The structure selected is correct for the most likely global event instantiations while the optimality of local structure is overlooked. Also, a large number of database events may be required the scheme to converges to the optimal structure since the number of possible network node instantiations may be large. It may also be noted that the data encoding length and the network encoding length are not normalized with respect to each other. While it may be possible add a scaling coefficient to one of the terms, it is not clear as to how this coefficient should be chosen without some prior knowledge of the network structure and database-feature statistical characteristics.

Finally, it has been shown that in the context of refining networks, using DGM as a heuristic is a viable and effective method for model selection in both the context of starting from a fully disconnected network as well as network refining. The DGM scoring function's strength comes from two qualities. First, DGM takes advantage of the directed Markov field property of BBNs in that when comparing BBN structures using DGM, the difference in score is determined by local structures only. Thus, by optimizing local structures the global network structure may be optimized. Second, DGM directly measures conditional expectations as well as their likelihood when determining whether or not to add an edge. It does not rely on indirect evidence, and this scheme of scoring the network in a manner similar to its use of showing statistical expectation seems to yield the best results. However, it is important to realize DGM heuristic structure learning does is not

as computationally efficient as other methods like the asymptotically correct information theory approach.

The Data Given Model probability provides an exceptional scoring heuristic when the main concern is deriving an accurate BBN. However, it is important to realize that DGM will however over-fit the BBN to the database if it is allowed to add edges unchecked. We see that the ability to add a threshold provides an advantage in that we may define how the narrowing of a context may provide an explanation. That is, we define quantitatively how much statistical expectation constitutes "causation" as opposed to correlation. However, it would be desirable to not need to define this value, and as mentioned before, it has been shown that causation can be shown by the use of a "virtual control." Future work in the area of finding a threshold could focus on attempting to use the showing of causality when deciding whether or not to add an edge.

We have also seen that we could prove that DGM is a convex function on a database if we can prove the simpler function $g(x_i, \pi_i)$ is itself a convex function on the database. Another area may focus on proving the convexity of $g(x_i, \pi_i)$ thereby proving that a global optimum may be found for a network given a database. If convexity cannot be proven, it may be beneficial to introduce constraints on the database which would induce a convex space for DGM on the database.

It may have been noted that in this thesis, only discrete BBN nodes were explored. For practical use, it cannot always be assumed that BBN nodes will be discrete. The use

of BBNs in areas such as computer vision will require the ability to deal with a mix of both continuous and discrete data dictating the need for not only parameter learning for continuous and discrete nodes, but also the interaction of different combinations of these nodes. Future work in this area should be concerned with abstracting the discrete node concepts presented in this thesis so that both discrete and continuous nodes can be handled within a common framework.

It may also have been noted that BBNs fall into a larger context of graph-based artificial intelligence structures. As noted before, BBNs are adept at modelling situations where there may be exceptions, doubt and a lack of regularity. There are other models, such as Markov chains that are adept at handling similar situations. Other models, such as support vector machines neural networks and fuzzy logic are adept at learning potentially high dimensional surfaces and quickly processing this information in a more causal context. We see that all of these models could be combined under a common framework with the capability of not only creating single type AI structures, but also mixed types thereby facilitating, among other things, the creation of sophisticated intelligent agents.

Finally, a fair amount of research has been done in the area of both BBNs as well as AI structures in general. Since many aspects of this area are still relatively new, more work in this area is expected. However, relatively little work has been done in actually using these structures, and it is many times through the exploration of applications that areas of interest are pushed further. For this reason, future work should consist of a placing a greater

emphasis on their applications.

Bibliography

[1] Bayesware Discovery software.
http://www.bayesware.com/frontpage.html

[2] J. Cheng, D. Bell, and W. Liu An algorithm for Bayesian network construction from data. *proceedings of the 6th Int'l. Workshop on AI and Stat*, 1997.

[3] J. Cheng. Belief Network (BN) PowerConstructor
http://www.cs.ualberta.ca/ jcheng/bnpc.htm

[4] J. Chen, D. Bell, and W. Liu. Learning belief networks from data: an information theory based approach. *proceedings of the Sixth ACM International Conference on Information and Knowledge Management*, 1997.

[5] R. Cowell, A. Dawid, S.Lauritzen, and D. Spiegelhalter. *Probabilistic Networks and Expert Systems*. Springer-Verlag, 1999.

[6] S. Friedland, and H. Schneider. Spectra of expansion graphs. *Proc. of the 11th Haifa Matrix Theory Conference*, pages 2–10, 1999.

[7] R. Graham, D. Knuth, and O. Patashnik *Concrete Mathematics*. Addisson-Wesley Publishing Company, 1994.

[8] I .Beinlich, H. Suermondt, R. Chavez, and G. Cooper. The ALARM monitoring system: A case study with two probabilistic inference techniques for belief networks. *Proc. of the Second European Conf. on Artificial Intelligence in Medicine*, pages 247–256, 1989.

[9] W. Lam and F. Bacchus. Learning Bayesian belief networks: An approach based on the MDL principle. *Computational Intelligence*, 10(4), 1994.

[10] M. Mitchell *An Introduction to Genetic Algorithms*. The MIT Press, 1999.

[11] K. Murphy. Bayes Net Toolbox for Matlab http://www.ai.mit.edu/ murphyk/Software/BNT/bnt.html

[12] K. Murphy. A Brief Introduction to Graphical Models and Bayesian Networks. http://www.ai.mit.edu/ murphyk/Bayes/bnintro.html.

[13] K. Murphy. Dynamic Bayesian Networks: Representation, Inference and Learning. PhD thesis, Dept. Comp. Sci., University of California, Berkeley., 2002.

[14] A. Pappas. The Accuracy of a Bayesian Network. http://www.doc.ic.ac.uk/deptechrep/DTR02-3.pdf.

[15] J. Pearl. *Causality: Models, Reasoning and Inference*. Cambridge Univ. Press, 2000.

[16] I. Rish, J. Hellerstein and T. Jayram. An analysis of naive Bayes classifier on low-entropy distributions. http://www.research.ibm.com/PM/ijcai01.ps.

[17] M. Ramoni and P. Sebastiani. Learning Bayesian Networks from Incomplete Databases. *Conference on Uncertainty in Artificial Intelligence*, pages 401-408, Morgan Kaufman, 1995.

[18] F. Sahin. *A Bayesian Network Approach to the Self-organization and Learning in Intelligent Agents*. PhD thesis, Dept. Electrical and Comp. Eng., Virginia Polytechnic and State University., 2000.